UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Mecânica

MARCUS VINÍCIUS PONTES LIMA

Reconstrução e Regularização de Trajetórias via Odometria Visual e Redes Neurais

CAMPINAS
2015

MARCUS VINÍCIUS PONTES LIMA

Reconstrução e Regularização de Trajetórias via Odometria Visual e Redes Neurais

Dissertação de Mestrado apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Mecânica, na Área de Mecânica dos Sólidos e Projeto Mecânico.

Orientador: Prof. Dr. Paulo Roberto Gardel Kurka

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO MARCUS VINÍCIUS PONTES LIMA E ORIENTADO PELO PROF. DR PAULO ROBERTO GARDEL KURKA

.....

ASSINATURA DO ORIENTADOR

PROF. DR. PAULO ROBERTO GARDEL KURKA
Matrícula 228842
FEM/UNICAMP

CAMPINAS
2015

Agência(s) de fomento e nº(s) de processo(s): FAPEAM, 002/2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

L628r

Lima, Marcus Vinícius Pontes, 1991-
Reconstrução e regularização de trajetórias via odometria visual e redes neurais / Marcus Vinícius Pontes Lima. – Campinas, SP : [s.n.], 2015.

Orientador: Paulo Roberto Gardel Kurka.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Redes neurais (Computação). 2. Blender (Programa de computador). 3. Fluxo óptico. 4. Visão por computador. 5. Navegação autônoma. 6. Processamento de imagens. I. Kurka, Paulo Roberto Gardel,1958-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Reconstruction e regularization of trajectories via visual odometry and neural networks
Palavras-chave em inglês:
Neural networks (Computer)
Blender (Computer program)
Optical Flow
Computer vision
Autonomous navigation
Image processing
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Paulo Roberto Gardel Kurka [Orientador]
Grace Silva Deaecto
Juliana Aparecida Fracarolli
Data de defesa: 26-11-2015
Programa de Pós-Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

DEPARTAMENTO DE MECÂNICA DOS SÓLIDOS E PROJETO MECÂNICO

DISSERTAÇÃO DE MESTRADO ACADÊMICO

Reconstrução e Regularização de Trajetórias via Odometria Visual e Redes Neurais

Autor: Marcus Vinícius Pontes Lima

Orientador: Prof. Dr. Paulo Roberto Gardel Kurka

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Paulo Roberto Gardel Kurka, Presidente
DSI/FEM/UNICAMP

Profa. Dra. Grace S. Deaecto
DMC/FEM/UNICAMP

Profa. Dra. Juliana A. Fracarolli
FEAGRI/UNICAMP

Campinas, 26 de Novembro de 2015.

DEDICATÓRIA

Dedico este trabalho à minha querida mãe.

AGRADECIMENTOS

Este trabalho teve colaboração de diversas pessoas, gostaria de prestar meus agradecimentos:

A Deus, por ser a esperança do impossível.

À minha mãe, pelo incentivo em todos os momentos da minha vida.

À minha querida namorada, por todo o apoio e por manter a minha vida em ordem.

Ao meu orientador, Prof. Dr. Paulo Kurka que foi além do seu papel de orientador e sempre me deu apoio quando necessário e confiou na minha capacidade para desenvolver este trabalho.

A todos os professores e colegas da UNICAMP, que contribuíram com seu tempo, conselhos, críticas e parte de suas pesquisas. Em especial, à Profa. Dra. Grace Deaecto por ter acreditado no meu potencial e sempre me dado conselhos valiosos, ao Prof. Dr. Ely Paiva pelos excelentes ensinamentos e por sempre estar presente quando necessário e à Profa. Dra. Juliana Fracarolli por ter gentilmente aceito participar na banca avaliadora e mostrar-se interessada na pesquisa.

A todos, muito obrigado.

“*No man is an island”*

(John Donne)

RESUMO

Aplicações na área de navegação e localização autônoma visual (vSLAM) utilizam sensores auxiliares à visão para diminuição dos erros de estimativa de posição. Neste trabalho, demonstra-se uma metodologia para a recuperação e a regularização de uma trajetória percorrida por um robô utilizando somente imagens estereoscópicas capturadas com uma determinada frequência durante sua movimentação. A simulação da movimentação e a captura de imagens são realizadas em um ambiente virtual desenvolvido na plataforma de computação gráfica 3D – Blender. A dinâmica de movimentação do robô, a recuperação da trajetória e a filtragem são implementados no MATLAB e o processamento de imagens é realizado no Visual Studio em C++ utilizando a biblioteca OpenCV.

Utilizando o algoritmo de processamento de imagens de Shi-Tomasi, são encontrados pontos de interesse em casa imagem. Com estes pontos e a técnica de fluxo óptico de Lucas Kanade, estima-se o deslocamento destes pontos em coordenadas de *pixel*. Utilizando relações geométricas, é possível converter essa movimentação para coordenadas do mundo, ou seja, recuperando a trajetória original do robô. Esta estimativa possui um erro de escala e rotação em relação à original; utiliza-se então um filtro projetado com redes neurais para melhoria dos resultados.

Palavras Chave: vSLAM; Imagens Esteroscópicas; Blender; Fluxo Óptico; Shi-Tomasi; Lucas Kanade; OpenCV; Robô.

ABSTRACT

Applications in the autonomous visual localization and mapping (vSLAM) field make use of auxiliary sensors to the vision in order to reduce the trajectory estimation error. In this work, it is presented a methodology to recover and regularize a trajectory of a robot using only stereoscopic images captured in a determined frequency during its path. The simulation of the robot's movement and the capture of images are done in a virtual environment designed in the 3D creation suite – Blender. The robot's dynamics, trajectory recovery and regularization (filtering) are implemented in MATLAB. Image processing is achieved in Visual Studio using C++ and OpenCV library.

Using the image-processing algorithm to find good points to track of Shi-Tomasi and the Optical Flow from Lucas Kanade, interest points are determined on the stereo image and their displacement is estimated in *pixel* coordinates. Using geometrical relations, it is possible to convert such displacement in world coordinates, i.e. recover the robot's original trajectory. Such estimation has a scale and rotation error in comparison to the original; a filter designed using neural network is applied for improvement.

Key Words: vSLAM; Stereo Images; Blender; Optical Flow; Shi-Tomasi; Lucas Kanade; OpenCV; Robot.

LISTA DE ILUSTRAÇÕES

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SLAM	*Simultaneous Localization and Mapping*
vSLAM	*Visual Simultaneous Localization and Mapping*
O.V.	Odometria Visual
GPS	*Global Positioning System*
GNSS	*Global Navigation Satellite System*
IMU	*Inertial Measurement Unit*
SINS	*Strapdown Inertial Navigation System*
ORB	*Oriented FAST and Rotated BRIEF*
SURF	*Speeded Up Robust Features*
SIFT	*Scale-invariant Feature Transform*
V.O.	*Visual Odometry*
SBA	*Sparse Bundle Adjustment*
RMS	*Root Mean Square*
SOHT	*Subset and Overset for Hysteresis Thresholds*
UTHT	*Unimodal thresholding for Hysteresis Thresholds*
WSSD	*Weighted Sum of Squared Differences*
SSD	*Sum of Squared Differences*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
SVD	*Singular Value Decomposition*
SSE	*Sum of Squared Errors*
SSW	*Sum of Squared Weights*
Nftool	*Neural Fitting Tool*
SVM	*Support Vector Machines*
KNN	"*K" Nearest Neighbours*

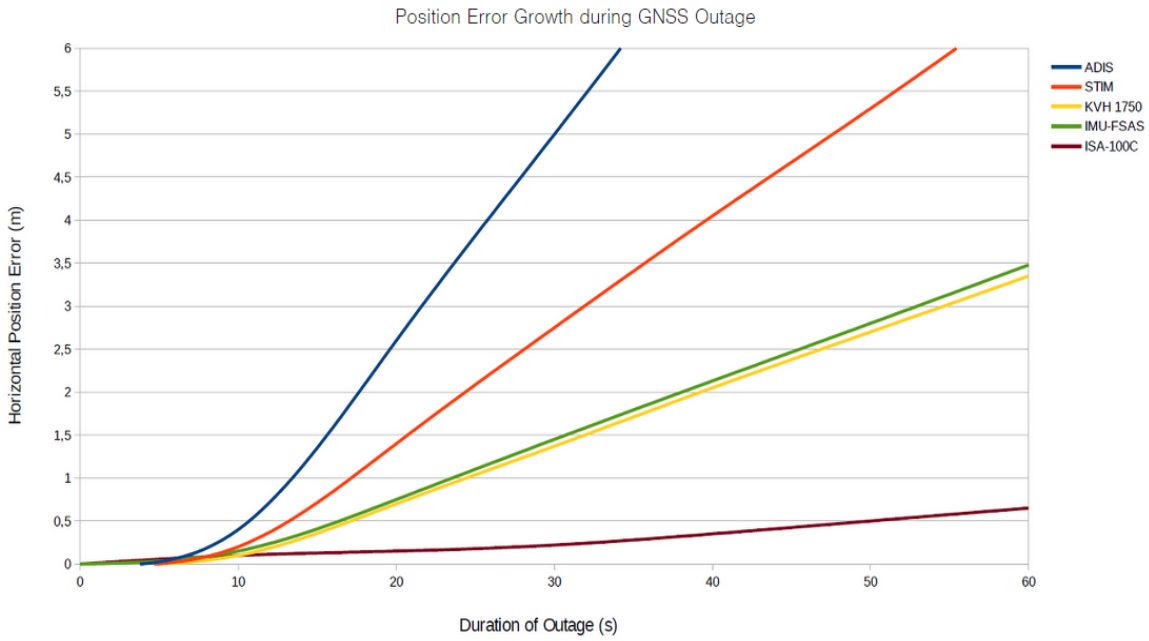
SUMÁRIO

1 INTRODUÇÃO

A localização e mapeamento simultâneos (SLAM[1]) é uma tarefa primária na navegação de robôs autônomos como visto em Weiss (2011). O uso de imagens para a realização de tal tarefa muda a abreviação para (vSLAM), onde o "v" refere-se a *visual*, neste caso, a estimativa de movimentação do robô é denominada odometria visual (O.V.). As aplicações nesta área englobam robôs terrestres Lategahn (2011), aéreos Fu (2014) e aquáticos Burguera (2014).

Mesmo em projetos atuais, verifica-se a utilização de sensores auxiliares à visão, como GPS, IMU, *encoders*, para tratar problemas de precisão presentes nos algoritmos de recuperação de trajetória por imagens. Informações de odometria de *encoders* rotacionais e sensores inerciais fornecem informações relevantes para a odometria mas estão sujeitos a problemas de deslizamento das rodas e acumulação de erros – tornando-as eficientes apenas para curtos movimentos em pequenos intervalos de tempo. A Figura 1.1 ilustra o efeito de *drift* presente na estimativa utilizando diferentes modelos de IMU. Cada cor identifica um modelo diferente do sensor.

Figura 1.1: Crescimento do erro de posição horizontal na utilização de diferentes IMUs em uma aplicação sem GNSS (*Gobal Navigation Satellite System*).



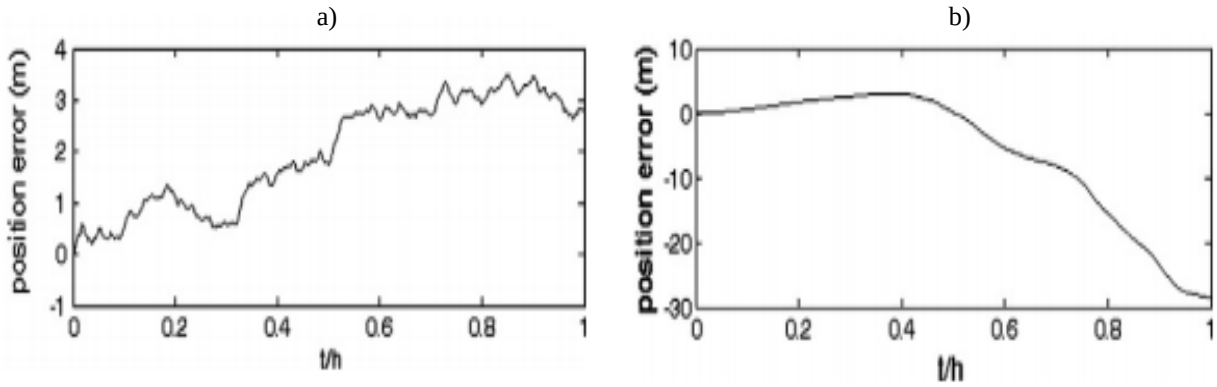
Fonte: Phoenix Aerial Systems (2014).

[1] Do inglês *Simultaneous Localization and Mapping*

A performance não está somente relacionada com o fabricante, mas também à tecnologia e os componentes internos do sensor. Fazendo com que o preço deste varie de U$15,00 (menor precisão) a U$ 31.000,00 (maior precisão) (Damien Douxchamps, 2015).

Trabalhos recentes utilizam GPS e SINS (*Strapdown Inertial Navigation System*) combinados para calcular a posição do robô. O SINS é composto por sensores de movimento (acelerômetros) e sensores de rotação (giroscópios). O trabalho de Zhang (2012) utiliza tais sensores e uma rede neural para estimar o posicionamento de um robô. A Figura 1.2 Ilustra um dos resultados do trabalho, destacando a diferença da precisão obtida com e sem GPS e SINS.

Figura 1.2: Erro de posição em metros vs. tempo (hora) com GPS (a) e sem GPS(b).



Fonte: Zhang (2012).

Mesmo com GPS, o método apresentou um erro de aproximadamente 3 metros em 1 hora e sem GPS um erro exorbitante de 30 metros em 1 hora. Tal técnica não seria adequada para um controle de trajetória.

Outro método bastante explorado é a fusão de Odometria Visual com IMU. Um trabalho de grande relevância nesta área é o de Konolige (2011). A tabela 1.1 ilustra o erro máximo e a média quadrática nas posições "XYZ". *Little Bit* e *Ft. Carson* são os locais de teste.

Tabela 1.1 Estatísticas dos erros da trajetória, em metros e porcentagem da trajetória.

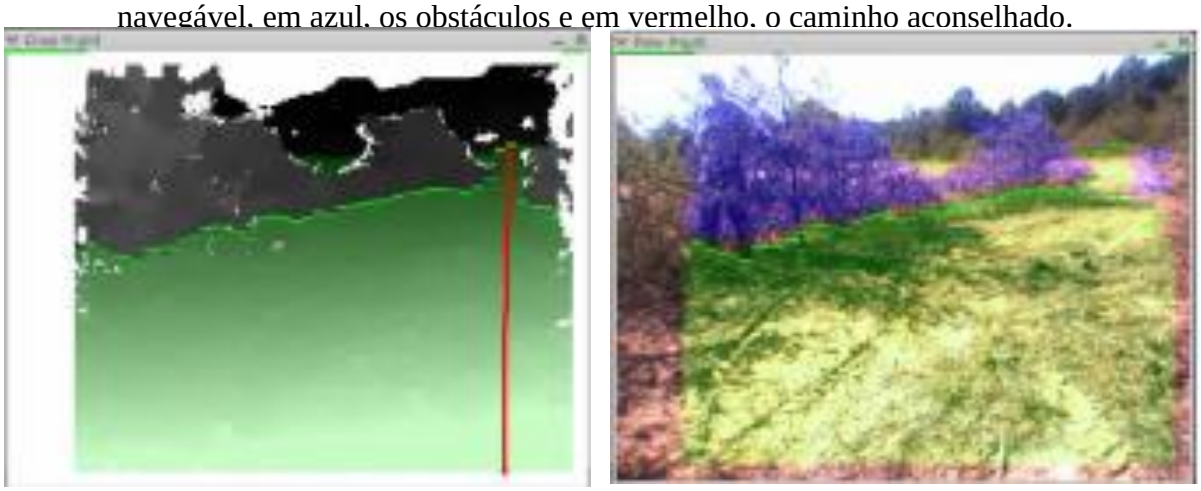
		RMS error in XYZ	Max error in XYZ
Little Bit	VO No SBA	97.41 (1.0%)	295.77 (3.2%)
	VO SBA	45.74 (0.49%)	137.76 (1.5%)
	VO No SBA + IMU	7.83 (0.08%)	13.89 (0.15%)
	VO SBA + IMU	4.09 (0.04%)	7.06 (0.08%)
Ft Carson	VO No SBA	263.70 (6.9%)	526.34 (13.8%)
	VO SBA	101.43 (2.7%)	176.99 (4.6%)
	VO No SBA + IMU	19.38 (0.50%)	28.72 (0.75%)
	VO SBA + IMU	13.90 (0.36%)	20.48 (0.54%)

Fonte: Konolige (2011).

Apesar do erro relativo em porcentagem ser pequeno, se for analisado em termos absolutos, o erro chega em até 295 metros utilizando apenas a odometria visual. Percebe-se então a necessidade de uma metodologia que consiga, apenas utilizando odometria visual, resultados bons o suficiente para serem aplicados em um controle de trajetória. Principalmente em locais onde a utilização de navegação por satélite não é possível.

Outra questão não abordada nos trabalhos citados é que a utilização de câmeras na odometria se torna muito vantajosa devido à grande quantidade de informações de podem ser retiradas de uma imagem. A Figura 1.3 mostra a aplicação de uma técnica de segmentação em uma imagem em um ambiente externo:

Figura 1.3: Imagem de disparidade (a) e Identificação de obstáculos (b). Em verde, a área navegável, em azul, os obstáculos e em vermelho, o caminho aconselhado.



a) b)

Fonte: Agrawal (2007).

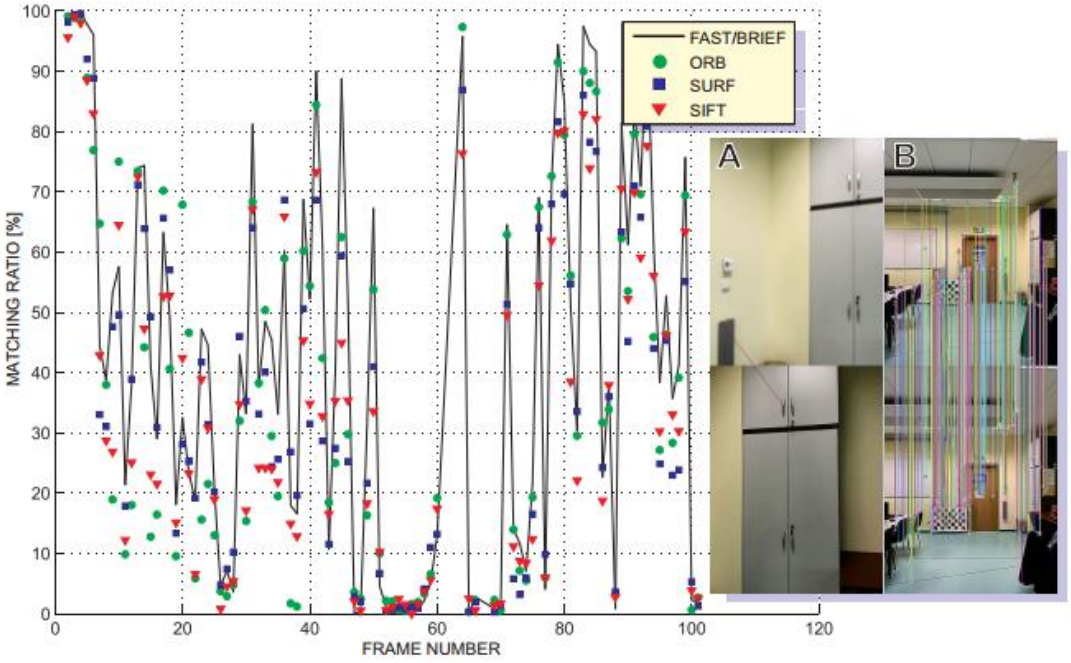
A segmentação é importante para distinguir áreas navegáveis de áreas proibidas para compor no algoritmo de planejamento de trajetória. Na Figura 1.3, poder-se-ia definir que o caminho indicado por uma reta vermelha seria uma possível trajetória indicada ao robô e a região em azul são os obstáculos que devem ser evitados.

Ademais, existem muitas outras técnicas que podem ser aplicadas como: realce, filtragem, representação, classificação; trazendo uma grande quantidade de detalhes que podem ser utilizadas na navegação do robô. Além disso, trabalhos como o de Kao (2013) utilizam a câmera presente em celulares com o sistema operacional *Android* para realizar o vSLAM, demonstrando que é possível, mesmo com uma câmera de baixo custo, realizar uma navegação visual em ambientes internos de maneira aceitável. Além disto, o trabalho de Nowicki (2014)

realiza a comparação de técnicas de detecção de pontos utilizando uma câmera de um dispositivo *Android*.

A Figura 1.4 ilustra a taxa de acerto de correlação entre *frames* utilizando uma câmera em um dispositivo com o sistema operacional *Android*. Em alguns *frames*, observa-se uma taxa de correlação acima de 80% dependendo do método utilizado – reforçando a possibilidade de uso de câmeras de baixo custo para aplicações em vSLAM. As cores representam diferentes técnicas de detecção de pontos de interesse na imagem.

Figura 1.4: Razão de acerto de correlação em relação ao número do frame das imagens de uma sala.



Fonte: Nowicki (2014).

O uso de câmeras como sensor para navegação móvel autônoma traz muitas vantagens em relação às outras alternativas. Primeiramente, tendem a ter um custo menor em relação aos sensores de alta precisão como *lasers*, sonar, GPS; em adição, câmeras são sensores de baixo consumo de energia por serem sensores passivos, ou seja, não precisam emitir nenhum tipo de sinal para obter as informações, apenas recebem a luz e a transforma em uma imagem digital.

Em adição, câmeras não dependem de uma estrutura externa, como o GPS, por exemplo – possibilitando aplicações em ambientes isolados e até em lugares não habitados (exploração planetária).

Apesar de ser um sensor versátil, o grande impacto que a câmera traz no processo de navegação está intimamente ligado com a parte de *software*, ou seja, a lógica (algoritmo) que

se utiliza para adquirir as informações. A grande questão desta interação é que se a programação for ineficiente, o processo pode ser tornar lento e acabar consumindo mais energia do que o esperado. Isto gera um grande desafio que é: como conseguir o máximo de informações com o mínimo de processamento possível?

1.1 Objetivo do Trabalho

O objetivo principal deste trabalho é apresentar uma metodologia para recuperação e regularização de trajetórias de robôs terrestres utilizando apenas câmeras como sensor e um filtro de redes neurais, levando aos seguintes objetivos específicos:

- Obtenção das imagens que descrevem o caminho do robô;
- Aplicar técnicas de processamento de imagem para adquirir os pontos de interesse;
- A cada passo de imagem, estimar o deslocamento dos *pixels* referentes aos pontos selecionados e armazenar os dados em um arquivo de texto;
- Realizar a reconstrução da trajetória e converter para coordenadas de mundo;
- Treinar a rede neural a partir de certas trajetórias pré-definidas;
- Utilizar a rede neural, previamente treinada, para regularização da trajetória; e
- Fazer um comparativo da trajetória real com a estimada pela odometria visual (O.V.) e a regularizada por redes neurais;

1.2 Estrutura do Trabalho

O presente trabalho é organizado do seguinte modo: os capítulos 2, 3 apresentam os principais fundamentos teóricos sobre o processamento de imagens, odometria visual e redes neurais. O Capítulo 4 discorre sobre a proposta de regularização de trajetórias. O capítulo 5 expõe o detalhamento do experimento realizado. O capítulo 6 discorre e ilustra os resultados obtidos. Por fim, no capítulo 7, apresentam-se as conclusões e sugestões para trabalhos futuros.

2 PROCESSAMENTO DE IMAGENS E ESTIMATIVA DE MOVIMENTO

A recuperação da trajetória original do robô através da O.V. é realizada a partir do cálculo de deslocamentos e rotações incrementais. Neste trabalho, tal processo é efetuado através do processamento das imagens estereoscópicas adquiridas por câmeras fixas ao robô. Dividindo-se esta tarefa em etapas, tem-se: detecção de pontos característicos, correlação dos pontos encontrados e estimativa da movimentação dos pontos em coordenadas de *pixel*. Tais etapas são descritas a seguir:

2.1 Detecção de Pontos Característicos

A detecção e a correlação de pontos em uma imagem são etapas cruciais para a odometria visual. Quanto melhor for a correlação dos pontos em duas imagens subsequentes, maior a precisão da estimativa de movimento e vice-versa.

Os pontos característicos e sua detecção possuem três principais vertentes: identificação de bordas, cantos e bolhas.

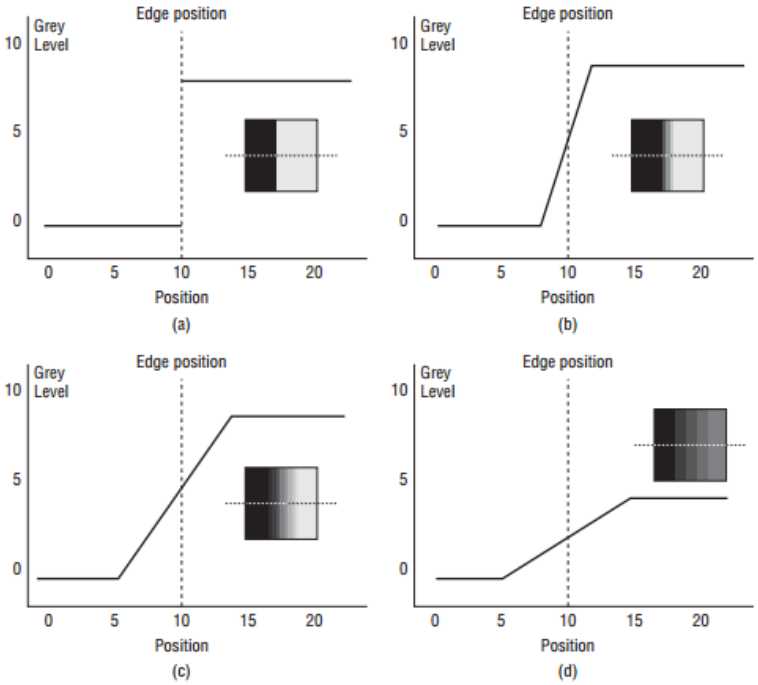
Existem diversas definições para caracterizar uma borda, cada uma podendo ser aplicada em diversas situações específicas. Uma das mais comuns e mais gerais definições é a borda degrau ideal, ilustrado na Figura 2.1.

A primeira complicação na caracterização de uma borda acontece devido à digitalização. É improvável que a imagem será amostrada de uma maneira tal que todas as bordas correspondam exatamente na fronteira de um *pixel*. De fato, a mudança de nível pode se estender através de um número de *pixels* (Figuras 2.1b, 2.1c e 2.1d). A atual posição do canto é considerada como sendo o centro da rampa conectando o nível mais baixo de cinza ao mais alto. Esta rampa é apenas uma representação matemática, já que depois que a imagem se torna digital (amostrada), a rampa tem uma aparência pontual de uma escada (Parker, 2011).

A segunda complicação é o problema, geralmente presente no mundo real, do ruído. Acontece devido a muitos fatores como a intensidade de luz, tipo de câmera e lentes, movimento e vibração, temperatura, efeitos atmosféricos, poeira, entre outros. É muito

improvável que dois *pixels* que correspondem precisamente ao mesmo nível de cinza na cena real terão o mesmo nível na imagem digitalizada. O ruído é um efeito aleatório e pode ser caracterizado apenas estatisticamente. O resultado do ruído em uma imagem é produzir uma variação aleatória no nível de intensidade de *pixel* a *pixel* e, portanto, uma transição suave das rampas ideais são muito improváveis em imagens reais.

Figura 2.1: Bordas-degrau.



Fonte: Parker (2011).

Na Figura 2.1(a), a mudança de nível ocorre exatamente no *pixel* 10. Em (b) A mesma mudança ocorre como em (a) mas durante 4 *pixels*, com o centro do degrau localizado no *pixel* 10. Em (c) e (d) ocorre similarmente a (b) mas sendo que o primeiro tem uma transição mais longa e o segundo uma mudança de nível menor.

Uma borda é definida como sendo uma mudança no nível de cinza, logo, um operador que é sensível a este tipo de mudança irá operar como um detector de bordas. Na literatura, os detectores de borda mais tradicionais são os de Sobel (1968), Canny (1986), e Harris(1988).

Considerando imagens bidimensionais (2D), é importante considerar mudanças de nível em diversas direções. Neste caso, as derivadas parciais de uma imagem são utilizadas em relação as direções principais *x* e *y*. Uma forma de se estimar a direção atual de um canto pode ser obtida utilizando as derivadas em *x* e *y* como componentes da direção atual ao longo dos eixos e computar a soma vetorial. O operador envolvido nesta situação é o gradiente.

Para uma imagem bidimensional *A* que é função de duas variáveis (*x*,*y*) , o gradiente é definido como:

$$\nabla A(x,y) = \left(\frac{\partial A}{\partial x}, \frac{\partial A}{\partial y}\right) \quad (2.1)$$

Onde $\partial A/\partial x$ e $\partial A/\partial y$ são as derivadas parciais da imagem. Como uma imagem é discreta, deve-se utilizar diferenças para o cálculo do gradiente; isto é, a derivada de um *pixel* é aproximada pela diferença de níveis de cinza sob uma região local. Uma possível aproximação seria:

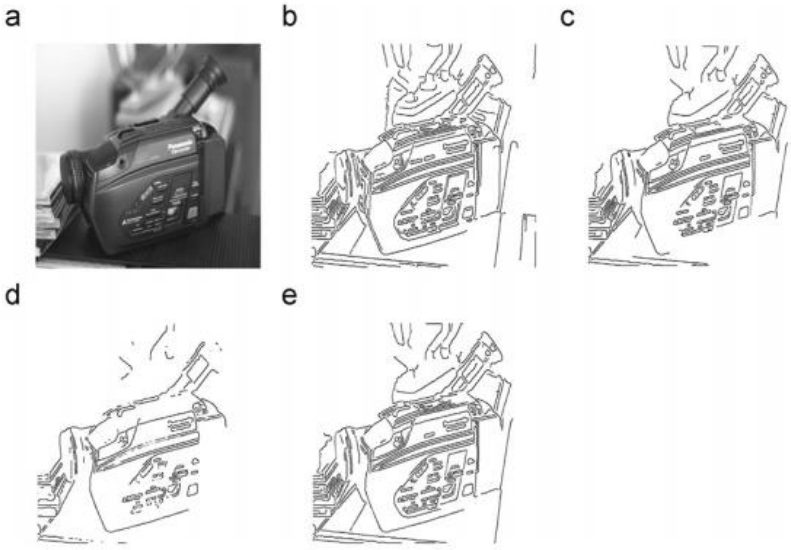
$$\nabla_{\mathrm{x}} A(x,y) = A(x+1,y) - A(x-1,y) \quad (2.2)$$

$$\nabla_{\mathrm{y}} A(x,y) = A(x,y+1) - A(x,y-1) \quad (2.3)$$

Este operador é simétrico com respeito ao *pixel* (*x*,*y*), apesar de não considerar o valor do *pixel* em (*x*,*y*). Independente do operador usado para computar o gradiente, o vetor resultante contém informações da intensidade da borda em um determinado *pixel* e qual a sua direção.

A Figura 2.2 ilustra o processo de detecção de bordas utilizando um método adaptado do método de Canny. O trabalho de Medina-Carnicer (2011) combina informações do gradiente com as informações obtidas por um processo de ligação aplicado a todos os candidatos.

Figura 2.2: Mapa de cantos obtido por diferentes métodos e a imagem de referência. (a) Imagem Original. (b) Imagem de referência. (c) mapa SOHT. (d) mapa UTHT. (e) mapa do método proposto.



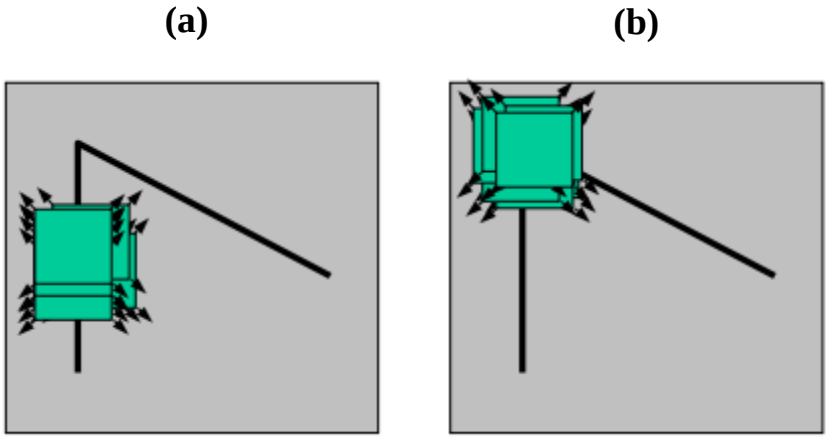
Fonte: Medina-Carnicer (2011).

A Figura 2.2a é a imagem de entrada, a Figura 2.2b ilustra o melhor resultado com parâmetros inseridos manualmente e as Figuras 2.2c, 2.2d e 2.2e são resultados obtidos automaticamente por diferentes métodos.

A detecção de um canto utiliza os mesmos princípios de gradiente aplicados nas bordas, entretanto, seguindo a ideia de Harris, na borda, não há mudança ao longo de sua direção e no canto, há uma mudança significativa em todas as direções. Detectores de cantos mais abordados em literaturas são os de Moravec (1977), Forstner (1986), Harris (1988), Shi-Tomasi (1994).

A Figura 2.3 ilustra como seria o processo de varredura na detecção de uma borda e de um canto. Em 2.3(a), percebe-se que se a varredura for feita ao longo da borda, não haverá mudança na intensidade desta. Já em 2.3(b), quando chega-se em um canto, haverá uma mudança significativa ao longo de todas as direções naquele determinado ponto.

Figura 2.3: Varredura ao longo de uma borda (a) e ao longo de um canto (b).



Fonte: Harris (1988).

Assumindo uma imagem em tons de cinza bidimensional A, toma-se uma área da imagem (u, v) e deslocando-a por (x, y). A soma ponderada dos quadrados das diferenças (*WSSD*[2]) é dado por:

$$S(x,y) = \sum_{u} \sum_{v} w(u,v)\big(A(u+x, v+y) - A(u,v)\big)^2 \qquad (2.4)$$

Onde $I(u + x, v + y)$ pode ser aproximado por uma expansão de Taylor. Adotando A_x e A_y como as derivadas parciais de I, tal que:

[2] Do inglês *Weighted Sum of Squared Differences*

$$A(u+x, v+y) \approx A(u,v) + A_x(u,v)x + A_y(u,v)y \tag{2.5}$$

Onde A_x e A_y são as derivadas parciais da imagem A. A equação 2.5 produz a aproximação:

$$\mathrm{S(x,y)} \approx \sum_{\mathrm{u}} \sum_{v} w(u,v)\big(A_x(u,v)x + A_y(u,v)y\big)^2 \tag{2.6}$$

A equação 2.6 pode ser escrita na forma matricial:

$$\mathrm{S(x,y)} \approx (\mathrm{x}\ \ \mathrm{y}) T_e \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.7}$$

Na equação 2.7, T_e é denominado tensor estrutural (*structure tensor*):

$$T_e = \sum_{\mathrm{u}} \sum_{v} w(u,v) \begin{bmatrix} A_x^2 & A_x A_y \\ A_x A_y & A_y^2 \end{bmatrix} \tag{2.8}$$

Tal matriz é conhecida como matriz de Harris. Um canto (em geral, um ponto de interesse) é caracterizado por uma grande variação de S (Eq. 2.7) em todas as direções do vetor $(x\ \ y)$.

Analisando os autovalores de T_e (λ_1 e λ_2), esta caracterização pode ser expressa da seguinte forma: O termo T_e deve ter dois autovalores de grande magnitude para um ponto de interesse. Baseado no valor absoluto dos autovalores, as seguintes inferências podem ser feitas:

1) Se $\lambda_1 \approx 0\ e\ \lambda_2 \approx 0$, então o *pixel* (x, y) não possui pontos de interesse.
2) Se $\lambda_1 \approx 0\ e\ \lambda_2$ for um valor alto positivo, então uma borda foi encontrada.
3) Se $\lambda_1\ e\ \lambda_2$ forem dois valores altos positivos, então um canto foi encontrado.

Os valores denominados altos positivos são relativos à cada imagem. Em uma região da imagem que contém bordas, serão observados os maiores valores encontrados.

Harris e Stephens notaram que o cálculo exato dos autovalores é computacionalmente caro, já que é necessário calcular uma raiz quadrada, então sugeriram a seguinte função:

$$\mathrm{R} = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(T_e) - \kappa trace^2(T_e) \tag{2.9}$$

Onde κ é um parâmetro de sensibilidade ajustável e R um valor utilizado no limiar.

O detector de cantos de Shi-Tomasi computa diretamente a função de limiar da seguinte forma:

$$R = \min(\lambda_1, \lambda_2) \tag{2.10}$$

Na equação 2.10, se λ_1, λ_2 são maiores que um valor mínimo λ_{min} (gatilho), o *pixel* é considerado um canto.

Outra importante característica são as bolhas. A detecção deste tipo de característica é desejável em muitas aplicações na biomedicina, como a identificação de nódulos em radiografias e contagem de células.

O trabalho de Liu (2010) propõe a detecção automática de bolhas por análise *Hessiana* e escalonamento. A Figura 2.4 ilustra um dos resultados do método proposto.

Figura 2-4: Detecção de bolhas na imagem (a) utilizando diferente métodos: (b) Frangi's. (c) Li's. (d) Método proposto por Liu.

Fonte: Liu (2010).

A detecção ilustrada na Figura 2-4 poderia servir para catalogar determinadas espécies de borboleta, por exemplo. Na robótica, as aplicações são inúmeras, podendo-se detectar peças com certo formato para manipulação com um robô *pick-and-place*, detecção de rostos, obstáculos e até mesmo detecção de bolas em competições de futebol robótico.

Uma bolha, é uma região de uma imagem tal que algumas propriedades são constantes ou variam pouco. Todos os pontos dentro de uma bolha podem ser considerados, em algum sentido, similares entre si.

Um dos primeiros e mais comuns detectores de bolhas é baseado no Laplaciano da Gaussiana. Dada uma imagem de entrada *A(x,y)*, esta imagem é convolucionada por um *kernel* Gaussiano:

$$\mathrm{g(x,y,t)} = \frac{1}{2\pi t^2} e^{\frac{-x^2+y^2}{2t^2}} \tag{2.11}$$

A uma certa escala *t* para uma representação no espaço de escalas:

$$\mathrm{L(x,y;t)} = g(x,y,t) * A(x,y) \tag{2.12}$$

Então o resultado de se aplicar o operador Laplaciano:

$$\nabla^2 L = L_{xx} + L_{yy} \tag{2.13}$$

é computado, normalmente resultando em respostas fortemente positivas para bolhas escuras de extensão $\sqrt{2t}$ e fortemente negativas para bolhas brilhantes de tamanho similar.

Um problema nessa abordagem é que quando aplica-se esse operador em uma única escala, a resposta se torna dependente da relação entre o tamanho da bolha no domínio da imagem e o tamanho do *kernel* Gaussiano usado para pré-suavização. Para trabalhar de forma automática, capturando bolhas de diferentes (desconhecidos) tamanhos no domínio da imagem, uma abordagem multi-escala é necessária. Uma forma mais direta para se obter um detector de bolhas multi-escala é considerar o operador Laplaciano normalizado em escala:

$$\nabla^2_{norm} L(x,y;t) = t(L_{xx} + L_{yy}) \tag{2.14}$$

2.2 Correlação de Regiões na Imagem

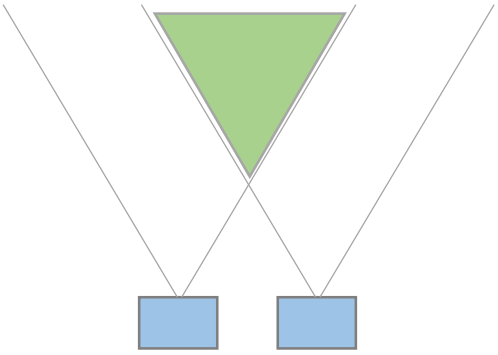
Em tarefas de visão que utilizam câmeras estereoscópicas ou necessitam estimar movimento, que é o caso neste projeto, encontrar características correspondentes entre duas imagens adjacentes e subsequentes é necessário.

Para isto, leva-se em consideração que a maioria dos pontos é visível em ambas as imagens (esquerda e direita) e que as regiões correspondentes das imagens são similares. É

preciso ter em mente que no caso de câmeras estereoscópicas, haverá sempre uma região de convergência no campo de visão e uma área exclusiva a cada câmera. A Figura 2.5 ilustra esta situação.

Observando a Figura 2.5, devemos considerar que os pontos obtidos em cada imagem devem estar dentro do campo de convergência. Garantindo assim, uma possível correlação entre os pontos obtidos.

Figura 2.5: Campo de visão de câmeras estereoscópicas e sua área de convergência (verde).



Fonte: Autor.

Um dos desafios na correlação é a busca de pontos na imagem esquerda que correspondam na imagem direita. Uma abordagem para tornar a busca mais rápida é utilizar uma varredura apenas na área onde o campo de visão das duas câmeras converge.

Além disso, a correlação de imagens de câmeras estereoscópicas pode ser estendida para análise de similaridades entre frames. Ou seja, a cada instante uma imagem é salva e então comparada com a anterior. Neste caso, se houve deslocamento, as imagens devem ter um fator de translação e de rotação que dificultam na correlação de pontos.

As técnicas a serem apresentadas nesta seção servem tanto para a correlação entre imagens estereoscópicas quando para imagens de uma mesma câmera mas em instantes diferentes.

Existem duas classes de algoritmos que são muito utilizados nessa tarefa: *correlation-based* e *feature-based*. No primeiro, é produzido uma grande quantidade de conjuntos de correspondência e no segundo, uma pequena quantidade. Neste trabalho, serão utilizados principalmente os algoritmos *correlation-based*. A tarefa então é encontrar regiões de similaridade em ambas as imagens (esquerda e direita). A figura 2.6 ilustra duas imagens similares e possíveis regiões de correlação.

Figura 2.6: Duas imagens obtidas por câmeras estereoscópicas.

Fonte: Collins (2012).

Na figura 2.6, escolhe-se arbitrariamente uma região na imagem da esquerda (quadrado vermelho). Deseja-se encontrar a mesma região na imagem direita, considerando diversas regiões selecionadas ao acaso (quadrados verdes), como encontrar a região correspondente?

Primeiramente, deve-se definir uma função que consiga, através da intensidade dos *pixels* das regiões, dizer se as áreas são correspondentes ou não. Ao mesmo tempo, através desta função deve-se encontrar um equilíbrio entre simplicidade e eficiência para não tornar o processo muito custoso computacionalmente.

Assumindo que a imagem da esquerda é uma função do tipo *A(x,y)* e a da direita *B(x,y)*, algumas possíveis medidas a serem consideradas são:

$$\max_{[x,y]\in R} |A(x,y) - B(x,y)| \tag{2.15}$$

$$\sum_{[x,y]\in R} |A(x,y) - B(x,y)| \tag{2.16}$$

$$\sum_{[x,y]\in R} \big(A(x,y) - B(x,y)\big)^2 \tag{2.17}$$

$$\sum_{[x,y]\in R} A(x,y)B(x,y) \tag{2.18}$$

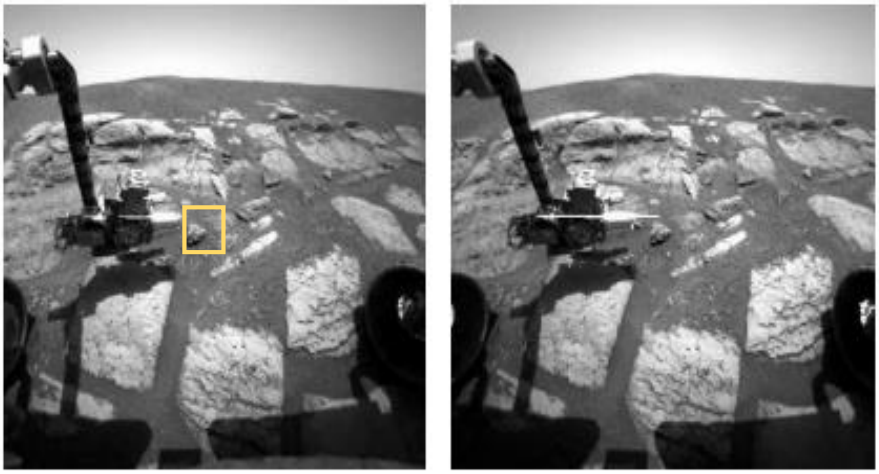
A equação 2.17 é uma das medidas mais utilizadas. É aplicada na obtenção de cantos pelo algoritmo de Harris. Pode-se criar uma relação entre as equações 2.17 e 2.18 da seguinte forma:

$$\sum_{[x,y]\in R} \big(A(x,y) - B(x,y)\big)^2 =$$

$$\sum_{[x,y]\in R} A(x,y)^2 + \sum_{[x,y]\in R} B(x,y)^2 - 2\sum_{[x,y]\in R} A(x,y)B(x,y) \qquad (2.19)$$

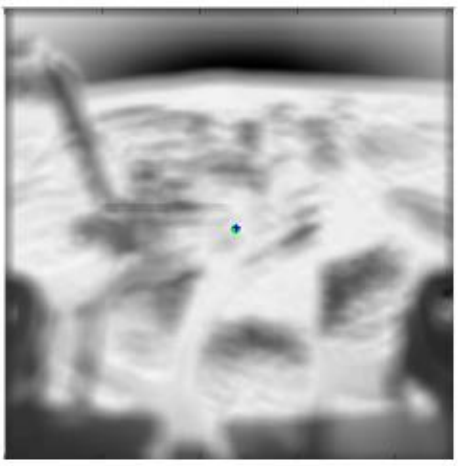
Supondo que deseja-se encontrar a correlação da região em amarelo na imagem da esquerda na imagem da direita. Se for aplicado a função de correlação da equação 2.17, conhecido como soma do quadrado das diferenças (*SSD*)[3], nas imagens da Figura 2.7 e então sobrepondo-as, tem-se o resultado da Figura 2.8.

Figura 2.7: Imagens estereoscópicas retiradas do *rover* da NASA em MARTE.



. Fonte: NASA (2012).

Figura 2.8: Correlação da região selecionada na imagem esquerda com a direita.



. Fonte: Collins (2012).

[3] Do inglês *Sum of Squared Differences*

Na Figura 2.8, percebe-se que foi possível encontrar uma correspondência bem sucedida para a região selecionada, ou seja, o maior índice de correlação na imagem coincide com a localização correta neste caso. Claramente nem sempre isto ocorrerá, muitas vezes é necessário customizar a função para atender necessidades específicas e considerar que a imagem pode sofrer alterações de iluminação, conter ruídos, estar distorcida e embaçada.

2.3 Estimativa do Movimento Utilizando Fluxo Óptico

O fluxo óptico, também conhecido como velocidade da imagem é utilizado em diversas aplicações, principalmente na área de acompanhamento de objetos, interpretação de cenas, reconhecimento de gestos, e navegação robótica.

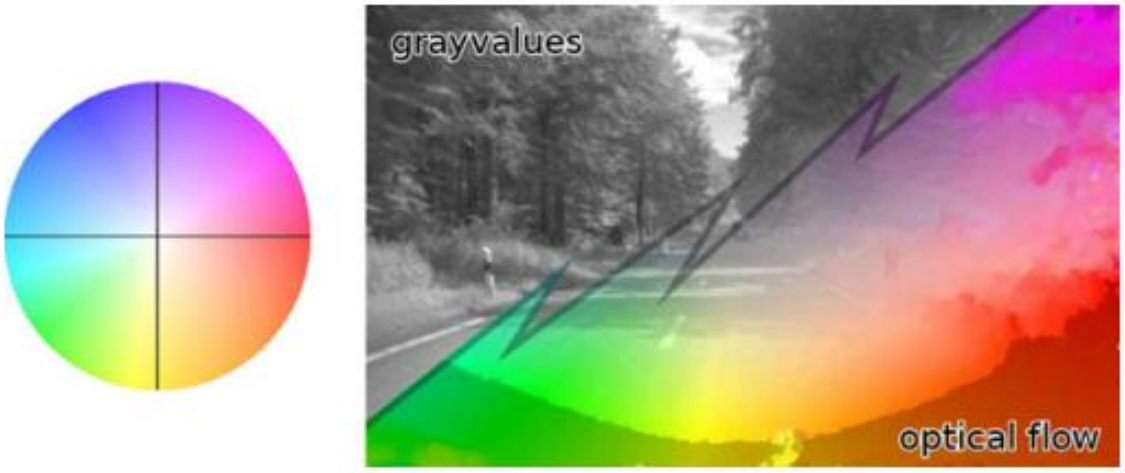
Se uma câmera ou um objeto na cena se move, este deslocamento resulta em uma função dependente do tempo descrevendo a mudança da intensidade dos *pixels* na sequência de imagens. O resultado bidimensional do campo de movimento aparente no domínio da imagem é chamado de campo de fluxo óptico.

Uma suposição comumente realizada na estimativa do fluxo óptico é a de constância de brilho, isto é, a intensidade de um determinado *pixel* correspondente em dois *frames* consecutivos deve ser a mesma. Infelizmente, nem sempre que um objeto se move há uma mudança na intensidade de *pixel* e nem sempre uma mudança na intensidade é gerada por uma movimentação de um corpo.

Para representar a direção de deslocamento dos *pixels* em um campo de fluxo óptico, existem duas maneiras mais utilizadas: codificação por cores e por setas. Na primeira, uma circunferência é dividida em quadrantes, cada um com uma cor diferente, indica a direção de deslocamento dos *pixels* de acordo com uma determinada cor, onde a intensidade da cor é maior nas bordas (indicando um maior deslocamento) e menor no centro, onde a cor branca representa que houve muito pouco ou nenhum deslocamento. A cor representa a direção dos *pixels* na região da imagem.

No segundo, um campo de setas é desenhado, onde a direção em que a seta está apontando é a direção que a determinada região da imagem está seguindo. Neste caso não há necessidade de diferenciar as direções por cores, já que a seta, independentemente da cor, já indica a direção de movimento. As Figuras 2.9 e 2.10 ilustram as duas representações.

Figura 2.9: Representação da imagem em escala de cinza e por fluxo óptico em cores.

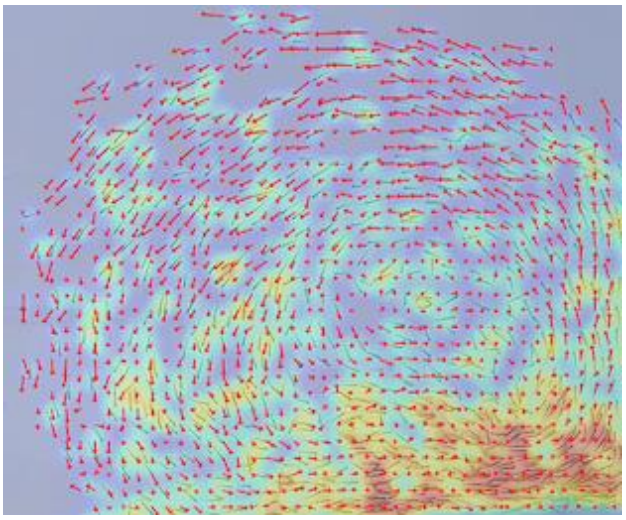


Fonte: Wedel (2011).

Na imagem 2.9, percebe-se que as cores do canto estão aumentando de intensidade de acordo com o quadrante a que pertencem. Isto indica que a movimentação da cena é para frente, pois os *pixels* estão se movimentando para fora da imagem.

Na imagem 2.10 se torna mais fácil de perceber a movimentação das regiões destacadas. Neste caso, o pesquisador estima a velocidade de um cardume de peixes.

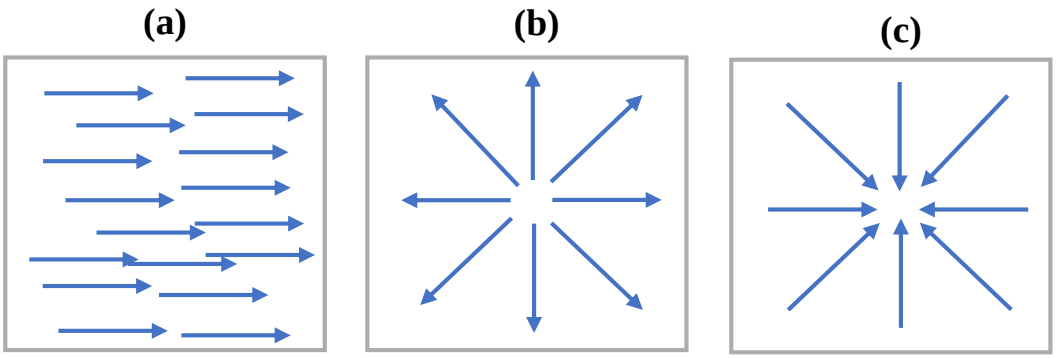
Figura 2.10: Representação do fluxo óptico por campo de cores e setas.



Fonte: Leblanc (2014).

Um detalhe importante é que os vetores de movimento do campo de fluxo óptico geralmente possuem direção oposta ao movimento da câmera. A Figura 2.11 ilustra alguns exemplos.

Figura 2.11: Representação do fluxo óptico por campo de cores e setas. (a) Câmera se deslocando para a esquerda; (b) Câmera se aproximando do centro. (c) Câmera se afastando do centro.



Fonte: Autor.

Para computar o fluxo óptico é apropriado trabalhar com uma imagem em tons de cinza. Desta forma, alguns cálculos são simplificados. A restrição do fluxo óptico considera que o valor de cinza de um *pixel* em movimento continua constante ao longo do tempo. Isto implica que:

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.20}$$

Onde *I(x,t,y)* é um conjunto de imagens amostradas em cada instante *t* , *u e v* são os incrementos de posição no instante *t+1*. A forma linearizada da versão da função de imagem (utilizando a aproximação de Taylor de primeira ordem) é dada:

$$I(x, y, t) \approx I(x + u, y + v, t + 1) + \nabla I(x, y, t + 1)^T \begin{pmatrix} u \\ v \end{pmatrix}$$

$$0 = I(x, y, t + 1) - I(x, y, t) + \nabla I(x, y, t + 1)^T \begin{pmatrix} u \\ v \end{pmatrix} \tag{2.21}$$

Onde T indica a ordem da derivada. Implicando na seguinte equação:

$$O(u, v): \quad I_t + I_x u + I_y v = 0. \tag{2.22}$$

Onde as derivadas parciais da imagem são denotadas como: I_t, I_x, I_y . Esta função do fluxo óptico implica um problema: uma equação para resolver um problema de duas variáveis (sistema subdeterminado). Sabe-se que um sistema de equações deste tipo possui um infinito número de soluções.

Uma maneira comum de se resolver esse problema de ambiguidade é assumir um campo de fluxo óptico constante em uma pequena vizinhança de um *pixel x*. Tal vizinhança $\mathcal{M}$ normalmente consiste de *n x n pixels* com n menor que 15. Esta condição é então avaliada com respeito a todos os *pixels* dentro desta janela $\mathcal{M}$. A operação agora resulta em um sistema com mais equações do que variáveis (sistema sobredeterminado).

Em geral, não existe nenhuma solução direta. Minimizar o quadrado da soma das derivações resulta na aproximação de mínimos quadrados, primeiramente proposto por Lucas e Kanade (1981):

$$\min_{u,v} \left\{ \sum_{x' \in \mathcal{M}(x)} \left(I_t(x') + I_x(x')u + I_y(x')v \right)^2 \right\} \tag{2.23}$$

Extensões dessa aproximação foram propostas. Incluindo a substituição do quadrado da soma dos erros com uma medida de erro absoluta ou utilizar filtros de Kalmann para futuros ganhos de robustez ao longo do tempo.

O fluxo vetorial resultante tem precisão em nível de *sub-pixel*, mas devido à aproximação de Taylor na função do fluxo óptico, este método é somente válido para vetores de pequenos deslocamentos onde o efeito em termo de maior ordem em 2.21 são desprezíveis.

Um modo de medir a qualidade do fluxo vetorial resultante é utilizar a soma da diferença dos valores de cinza dentro da vizinhança $\mathcal{M}$. Se o valor for maior que um valor pre-determinado (gatilho), pode significar que mais iterações são necessárias ou o processo ficou parado em um mínimo local.

Atualmente, o método de Lucas Kanade está além de uma aplicação de tempo real se o vetor de fluxo é estimado para cada *pixel* da imagem de entrada utilizando uma implementação com uma CPU comum. Aplicações recentes baseadas em GPU são capazes de rastrear até 10.000 pontos de imagem a cadência de 25 Hz (Wedel, 2011).

Neste trabalho, o fluxo óptico é utilizado para estimar a direção dos pontos característicos encontrados na imagem. O rastreamento apenas de alguns pontos pré-selecionados funciona como um filtro, onde os pontos mais relevantes são escolhidos e com a aplicação do fluxo óptico, pode-se então estimar o deslocamento destes pontos em coordenadas de *pixel*.

3 REGULARIZAÇÃO ATRAVÉS DE REDES NEURAIS ARTIFICIAIS

Redes neurais são complexos modelos não-lineares construídos com componentes que, individualmente, se comportam como um modelo de regressão. Apesar da estrutura de uma rede neural ser explicitamente desenvolvida antes, o processamento que a rede neural realizada para produzir a hipótese evolui durante o processo de aprendizado. Isto permite a rede neural a ser utilizada como um solucionador auto-programado, diferente de típicos algoritmos que devem ser desenvolvidos e codificados explicitamente (Cowan, 2013).

3.1 Estrutura da Rede Neural

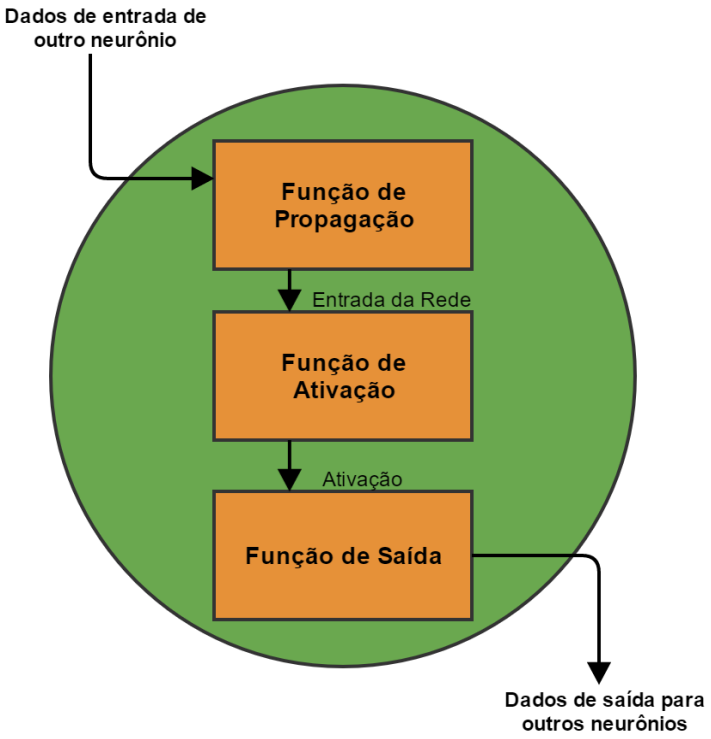
Uma rede neural é composta de neurônios, conexões e pesos. Os pesos podem ser implementados em uma matriz quadrada de pesos *W* ou em um vetor com o número da linha indicando onde a conexão inicia e o número da coluna indicando qual é o neurônio alvo. A Figura 3.1 ilustra o processamento de dados em um neurônio. Dados são transferidos entre neurônios através de conexões com o peso conectado sendo excitatório ou inibitório.

Observando um neurônio *j*, normalmente encontram-se diversos neurônios conectados a este, logo, transferindo suas saídas para *j*. Para este neurônio *j*, a função de propagação recebe as saídas $o_{i1}, \dots, o_{in}$ de outros neurônios $i_1, i_2, \dots, i_n$ (conectados a *j*) e as transforma de acordo com os pesos conectados para dentro da entrada da rede (*network input)* para poder ser futuramente processada pela função de ativação. Portanto, a entrada da rede é o resultado da função de propagação. Uma função de propagação bastante utilizada é a soma ponderada, isto é:

$$\sum_{i \in I} (o_i \cdot w_{i,j}) \tag{3.1}$$

Onde $w_{i,j}$ são os pesos de cada neurônio *i* conectado ao neurônio *j e I* um conjunto de neurônios $\{i_1, i_2, \dots, i_n\}$.

Figura 3.1: Processamento de dados em um neurônio. A ativação de um neurônio implica no valor de gatilho.



Fonte: Autor.

A ativação é o estado alternante de um neurônio. Considerando um neurônio *j*, seu estado de ativação a_j, em uma ativação curta, é explicitamente assinalado a *j*, indicando a extensão da atividade de um neurônio e resulta da função de ativação.

Perto do valor de gatilho (*threshold*), a função de ativação de um neurônio reage particularmente de modo sensível. O valor deste gatilho normalmente é definido na função de ativação. Cada neurônio pode ter seu próprio valor de gatilho que normalmente define o valor de máximo gradiente da função de ativação.

Diferentemente de outras variáveis dentro da rede neural, a função de ativação, também conhecida como função de transferência, normalmente é definida globalmente para todos os neurônios ou pelo menos para um conjunto de neurônios onde somente os valores de gatilho são diferentes de cada neurônio. Os valores de gatilho podem ser modificados por um processo de aprendizagem (*learning*). Em alguns casos, pode-se relacionar este valor como uma função no tempo. Uma função de ativação muito comum é a função de *Fermi* ou *função logística*:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

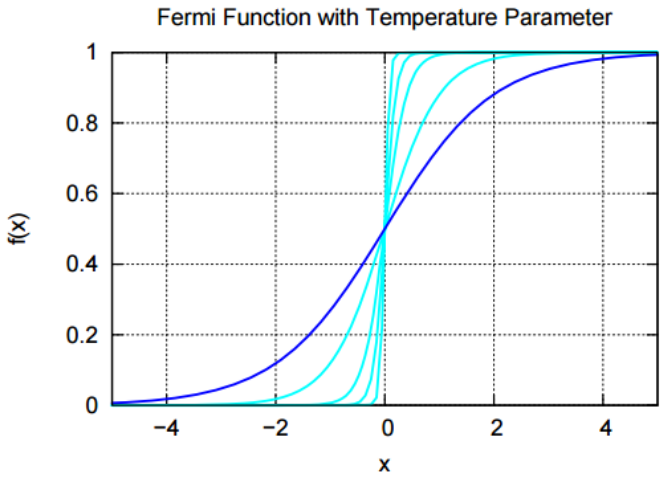
Tal função mapeia valores no intervalor de (0 a 1). Esta mesma função pode ser parametrizada na seguinte forma:

$$f(x) = \frac{1}{1 + e^{-\frac{x}{T}}} \tag{3.3}$$

Onde T é um parâmetro variável. Quanto menor for o valor de T mais a função é comprimida, ou seja, apresenta uma transição menos suave até se aproximar de uma função degrau, como visto na Figura 3.2.

Existem outras funções de ativação que não são explicitamente definidas mas dependem de uma entrada de acordo com uma distribuição aleatória (funções de ativação estocásticas).

Figura 3.2: Função de *Fermi* parametrizada.



Fonte: Kriesel (2005).

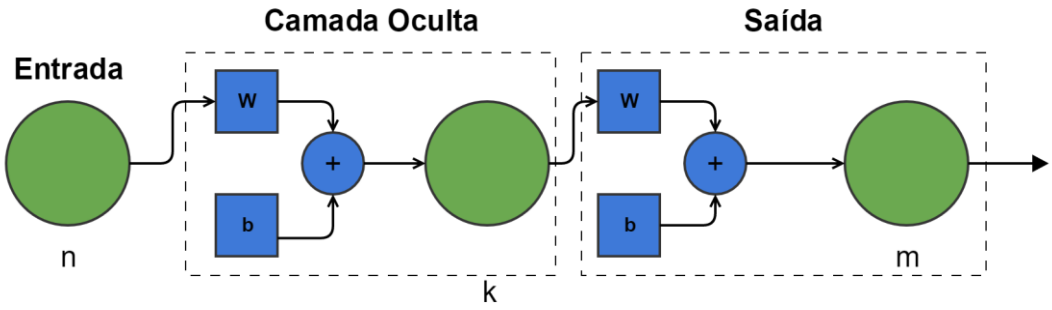
Com isso, pode-se definir o conceito de hipótese em redes neurais – um conjunto de dados de entrada aplicados a uma função de transferência ótima não-linear. Isto é representado na seguinte equação:

$$h_\theta(\vec{x}) = g(\vec{x}\theta) \tag{3.4}$$

Onde θ é a matriz definindo a transformação linear dos parâmetros de entrada, x é um vetor coluna de entradas e $g(k)$ é a função de transferência não-linear.

Um outro componente que pode ser adicionado à rede neural é o *bias* (viés) que é um valor constante, podendo ser diferente para cada neurônio, adicionado a fim de se realizar uma compensação. A Figura 3.3 ilustra a estrutura de tal rede.

Figura 3.3: Modelo da Rede Neural com pesos (w) e *biases* (b).



Fonte: Autor.

3.2 Aprendizado e Treinamento da Rede Neural

Um processo essencial para a eficiência de uma rede neural é o aprendizado. Este processo é movido por um conjunto de "m" exemplos, ou seja, "m" valores de $\vec{x}$ (entrada) para quais os valores correspondentes de *y* (saída) são conhecidos. O conjunto de valores de $\vec{x}$ podem ser agrupados em uma matriz X e os valores correspondente valores de *y* em uma matriz Y.

A hipótese[4] então se torna:

$$h_\theta(X) = g(X\theta) \tag{3.5}$$

O processo de aprendizagem pode ser agora definido como um processo que minimiza a função de custo:

$$J_\theta(\varepsilon), \text{ onde } \varepsilon = \left|\left|h_\theta(X) - Y\right|\right|^2 \tag{3.6}$$

Esta função de valores não negativos fornece algumas indicações da precisão da atual hipótese h_θ no conjunto de dados [Y, X] e o objetivo da etapa de aprendizado é minimizar o valor de J_θ.

No improvável evento de que a matriz X for quadrada e não-singular, a inversa pode ser usada para encontrar θ. Para outros casos, a pseudo-inversa pode ser utilizada:

$$Y = X\theta \rightarrow \theta = (X^T X)^{-1} X^T Y \tag{3.7}$$

[4] Hipótese pois o resultado é apenas uma suposição gerada a partir da transformação dos dados de entrada.

Esse procedimento minimiza $\left||X\theta - Y|\right|^2$ ou seja, assuma que $h_\theta(X) = X\theta$. Já que a matriz a ser invertida é simétrica e tipicamente semi-definida positiva, pode-se utilizar a decomposição em valores singulares (S.V.D.[5]).

Aproximações sucessivas do mínimo da função de custo podem ser obtidas através do gradiente descendente:

$$\theta_{i+1} = \theta_i - \alpha\vec{\nabla}J_\theta \tag{3.8}$$

Onde α é a taxa de aprendizagem que define a velocidade com que o algoritmo de aprendizagem converge. Um α pequeno irá resultar em um aprendizado lento, enquanto um maior valor de α resultará em oscilações em torno de um mínimo, impedindo a convergência.

A função com decaimento regular pode ser utilizada no lugar da constante α para fornecer uma rápida convergência em direção a um mínimo mas também reduzir a distância final do mínimo se o algorítmo oscilar.

Se as características no vetor de entradas possuírem diferentes escalas, um processo de normalização pode ser aplicado da seguinte maneira:

$$x_{norm} = \frac{x - \mu}{\sigma} \tag{3.9}$$

Onde μ é a média e σ o desvio padrão de x. Na equação 3.9, a média e o desvio padrão de uma lista *x* contendo *N* valores são respectivamente:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.10}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3.11}$$

O método de aprendizado abordado minimiza o resíduo $\left||h_\theta(X) - Y|\right|$, mas consequentemente ajusta o ruído e erros nos dados de aprendizado também. Para prevenir isto,

[5] Do inglês *Singular Value Decomposition*

outros termos podem ser adicionados no processo de minimização. Um exemplo é a adição da magnitude dos valores de aprendizado θ, modificando o resíduo para:

$$\left|\left|h_\theta(X) - Y\right|\right|^2 + \lambda\left|\left|\theta\right|\right|^2 \tag{3.12}$$

A variável λ, neste caso, é o parâmetro de regularização e determina quando o processo de aprendizagem minimiza a magnitude do vetor de parâmetros θ, que resulta em subajuste (minimizando λ) ou quando o processo realiza um sobreajuste dos dados de treinamento (X ,Y) (aumentado λ).

Existem diversos algoritmos de aprendizagem, os principais são: *Levenberg-Marquardt* de Moré (1978), *Bayesian Regularization* de MacKay (1992) e *Scaled Conjugate Gradient* de Møller (1993). Todos atualmente estão implementados no *toolbox* de redes neurais do MATLAB.

Neste trabalho, a regularização ou interpolação Bayesiana é a mais adequada pois obtém bons resultados com conjuntos de dados pequenos e ruidosos – presentes em aplicações de processamento de imagens onde possam existir ruídos de iluminação e vibração.

A regularização Bayesiana é realizada com a soma do quadrado dos erros e é modificada para incluir um termo que consiste de uma soma de quadrados dos erros e pesos da rede:

$$F_{reg} = \beta \cdot SSE + \alpha \cdot SSW \tag{3.13}$$

SSE e SSW são dados por:

$$SSE = \sum_{q=1}^{N} e_q^2(x) \tag{3.14}$$

$$SSW = \sum_{j=1}^{n} w_j^2 \tag{3.15}$$

Onde n e N são o número total de *biases* e pesos respectivamente, w_j são os valores do pesos na rede e e_q os erros em cada posição da entrada. O índice de performance em 3.12 força os pesos e as *biases* a serem pequenas, o que produz uma resposta mais suave e evita sobreajuste. Os valores de α e β são valores determinados de maneira automática (interpolação).

4 PROPOSTA DE REGULARIZAÇÃO DE TRAJETÓRIAS ESTIMADAS

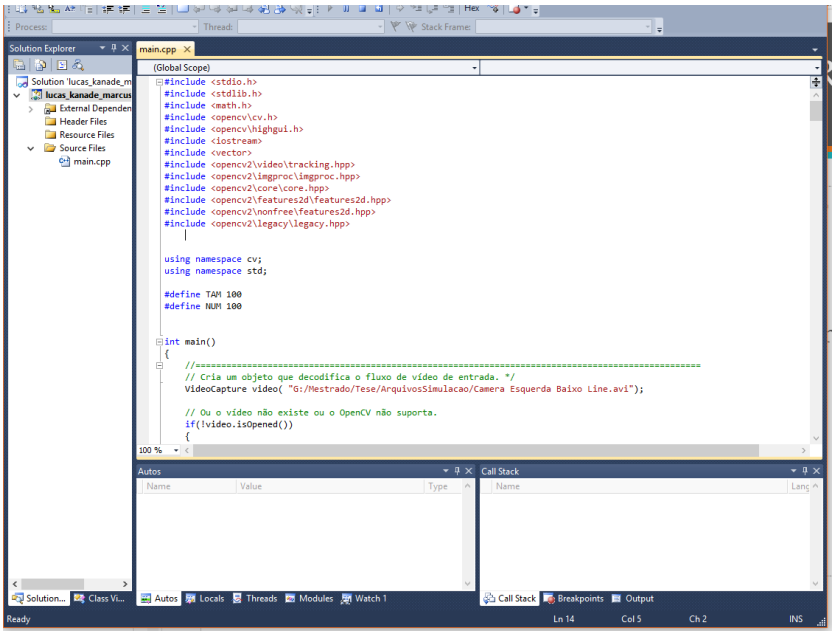
A Proposta de regularização de trajetórias estimadas pode ser dividida em quatro etapas principais:

- Aplicação da detecção de pontos e do fluxo óptico em um banco de imagens ou vídeo descrevendo a trajetória do robô;
- Conversão das coordenadas de *pixel* para coordenadas de mundo;
- Utilização de câmeras estereoscópicas para reconstrução do movimento do robô;
- Regularização por Redes Neurais;

4.1 Aplicação da Detecção de Pontos e do Fluxo Óptico

A aplicação do algoritmo de Fluxo Óptico será implementada na linguagem de programação C++ utilizando o ambiente de desenvolvimento *Visual Studio* e a biblioteca de processamento de imagens *OpenCV*. A Figura 4.1 Ilustra a interface deste ambiente.

Figura 4.1: Ambiente de desenvolvimento *Visual Studio 2010*.



Fonte: Autor.

A aplicação utilizará como entrada um banco de imagens ou vídeos descrevendo a trajetória do robô. Um detalhe importante é que as imagens devem estar indexadas de forma a respeitar a movimentação do robô a cada passo. Se as imagens estiverem embaralhadas ou o vídeo corrompido a estimativa não poderá ser realizada com precisão.

Após a identificação do vídeo descrevendo a trajetória do robô, deve-se definir as dimensões da máscara de busca de pontos característicos. Só poderão ser encontrados pontos que estão dentro da área de busca. Esta máscara não pode ser maior que a imagem e deve ter pelo menos 50% de área em relação à área total da imagem. A Figura 4.2 Ilustra a aplicação de tal máscara.

Figura 4.2: Aplicação de uma máscara para definição da área de busca dos pontos característicos.

Fonte: Autor.

Após da definição da região de busca, aplica-se a técnica de Shi-Tomasi para encontrar pontos característicos na imagem. A função que aplica tal técnica, utilizando a biblioteca *OpenCV*, tem a seguinte estrutura:

Tabela 4.1: Função *goodFeaturesToTrack* (Shi-Tomasi) e seus parâmetros.

	goodFeaturesToTrack							
Parâmetros	Imagem	Var. Saída	(int) Num. máximo de cantos	(double) Nível de Qualidade	(double) Distância Mínima	Máscara	(int) Tamanho do Bloco	(double) *k* (Parâmetro de Sensibilidade)

Fonte: Autor.

- Imagem: Variável de entrada do tipo matriz que contém os dados da imagem;
- Var. Saída: Variável onde são armazenadas as posições dos pontos encontrados;
- Num. Máximo de Cantos: Variável do tipo inteiro que limita a quantidade de cantos a serem encontrados;
- Nível de Qualidade: Variável do tipo *double* que define um parâmetro para qualidade dos pontos a serem encontrados. Quanto menor este valor, melhores os pontos, porém restringe a busca podendo encontrar um número menor de pontos;
- Distância Mínima: Variável do tipo *double* que define a mínima distância Euclidiana possível entre os cantos a serem encontrados;
- Máscara: Variável criada anteriormente que define a região de busca;
- Tamanho do Bloco: Variável do tipo inteiro que define o tamanho do bloco para computar a matriz de covariação derivativa em cada vizinhança do *pixel*; e
- k: Parâmetro de Sensibilidade do detector de Harris;

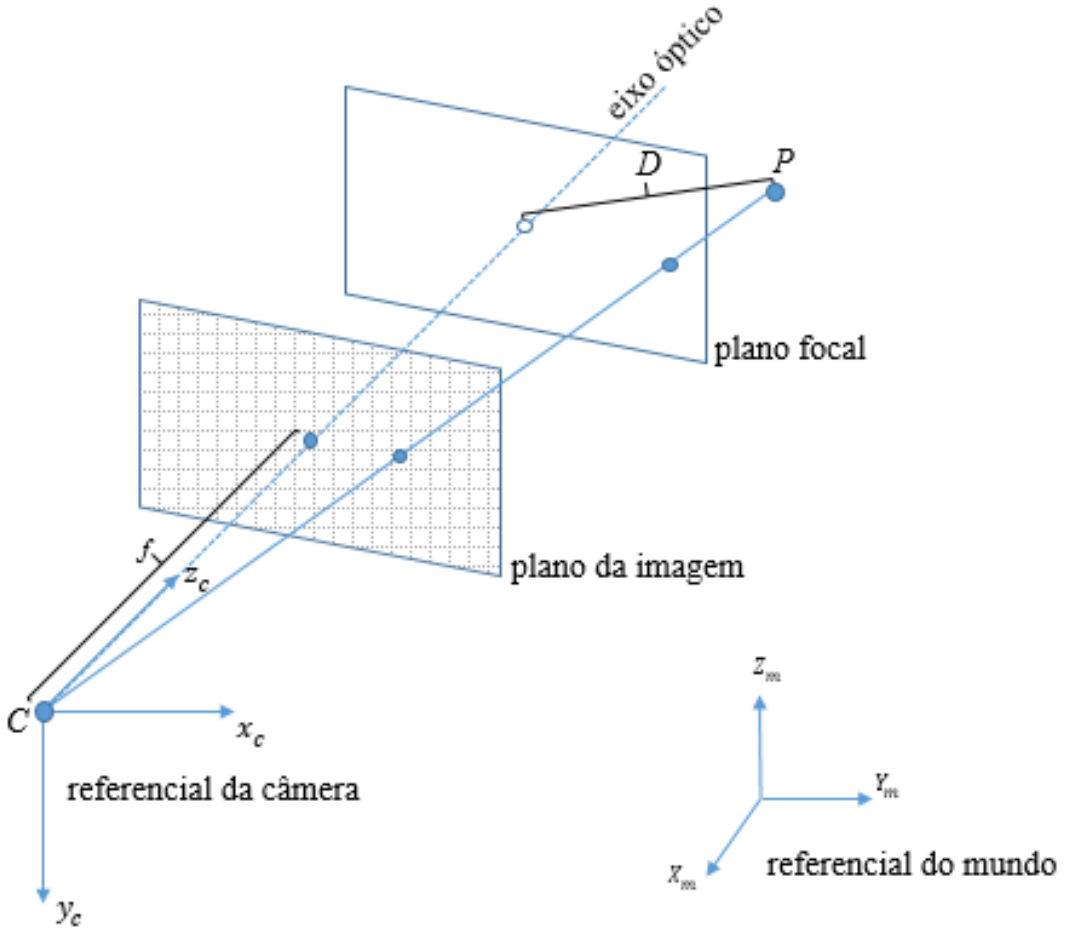
Após a obtenção dos pontos de interesse, como o interesse é recuperar a trajetória original de um robô, deseja-se realizar uma conversão de coordenadas de *pixel* para coordenadas de câmera e então para coordenadas de mundo. Este processo é realizado através de geometria epipolar.

4.2 Conversão de Coordenadas e Composição da Trajetória do Robô

Levando em consideração uma câmera do tipo *pinhole* representado na Figura 4.3, onde P é o objeto no mundo, D é a distância no orifício da câmera ao objeto, f é a distância focal, C representa o centro do referencial da câmera, x_c, y_c, z_c coordenadas do ponto no referencial de câmera e X_m, Y_m, Z_m coordenadas do ponto em referencial de mundo.

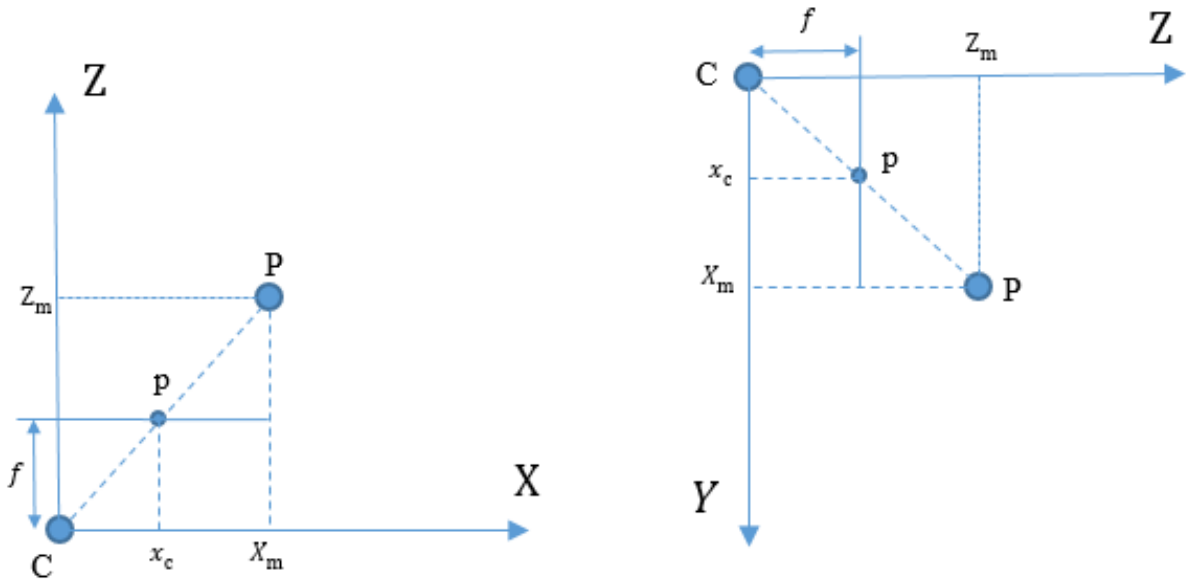
As imagens capturadas pela câmera são projetadas no plano imagem de forma perspectiva, ou seja, em cada ponto do objeto real fotografado, traça-se uma reta imaginária até a origem do sistema de coordenadas da câmera e essa linha intercepta o plano imagem, logo, este ponto de intersecção é o que forma a imagem no espaço bidimensional. A Figura 4.4 ilustra a projeção perspectiva de um ponto qualquer visto pela câmera.

Figura 4.3: Modelo de câmera *pinhole*.



Fonte: Autor.

Figura 4.4: Projeção perspectiva de uma câmera *pinhole*.



Fonte: Autor.

Onde, na Figura 4.4, p se refere ao ponto do mundo em coordenadas de câmera. Através de semelhança de triângulos, as seguintes relações são definidas:

$$\frac{X_m}{Z_m} = \frac{x_c}{f} \quad e \quad \frac{Y_m}{Z_m} = \frac{y_c}{f} \tag{4.1}$$

Onde X_m, Y_m e Z_m são as coordenadas de mundo, f é a distância focal, x_c, y_c e z_c as coordenadas no referencial da câmera. Isolando as coordenadas de câmeras obtém-se:

$$x_p = f\frac{X_m}{Z_m} \quad e \quad y_p = f\frac{Y_m}{Z_m} \tag{4.2}$$

Vetorialmente, pode-se representar P (mundo) e p (câmera) da seguinte forma:

$$P = \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix} \quad e \quad p = \begin{bmatrix} x_p \\ y_p \\ f \end{bmatrix} \tag{4.3}$$

Isto implica que:

$$p = \frac{f}{Z_m} \cdot P \tag{4.4}$$

Levando em consideração que no processo de odometria visual, primeiramente são encontrados pontos na imagem, coordenadas *(x,y)* em *pixels*. Precisa-se convertê-los para coordenadas de câmera para então obter as coordenadas de mundo. A relação entre o sistema de unidades métricas e o sistema de unidades de imagem (*pixels*) é dada por dois parâmetros intrínsecos da câmera, α_x e α_y, dados por:

$$\alpha_x = \frac{N_x}{L_x} \quad e \quad \alpha_y = \frac{N_y}{L_y} \tag{4.5}$$

Onde N_x e N_y são, respectivamente, as dimensões horizontal e vertical do plano de imagem dadas em *pixels* (resolução), L_x e L_y seus respectivos comprimentos em unidades métricas.

As coordenadas de imagem correspondentes a um ponto com projeções métricas (x_c, y_c) na câmera são dadas pela equação 4.6

$$u_p = \alpha_x x_c + u_0 \quad e \quad v_p = \alpha_y y_c + v_0 \tag{4.6}$$

Onde u_p *e* v_p são as coordenadas do ponto em questão em *pixels* e u_0 e v_0 as coordenadas da origem do sistema em *pixels*. A equação 4.6 pode ser escrita na forma vetorial utilizando a equação 4.3 da seguinte maneira:

$$\hat{p} = \frac{1}{f} A \cdot p \tag{4.7}$$

Onde $\hat{p}$ são as coordenadas do ponto na imagem e A a matriz de parâmetros intrínsecos:

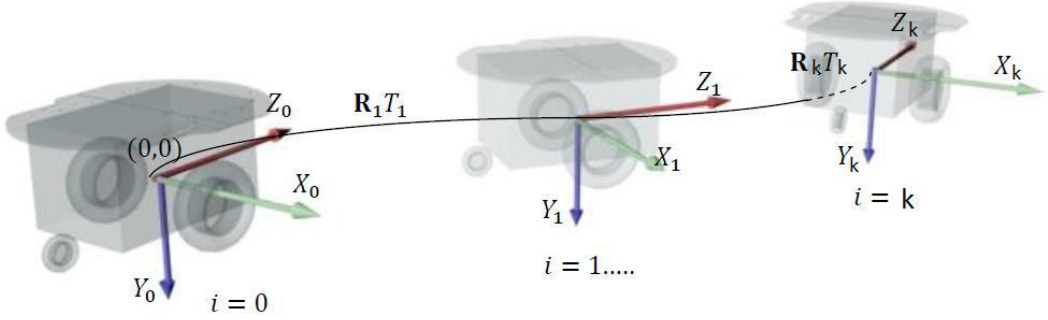
$$\hat{p} = \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} \quad e \quad A = \begin{bmatrix} \alpha_x f & 0 & u_o \\ 0 & \alpha_y f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

Pode-se então, tomando por base a equação 4.7, converter pontos em *pixels* para pontos em coordenada de câmera fazendo:

$$p = f A^{-1} \cdot \hat{p} \tag{4.9}$$

Para compor a movimentação do robô, precisa-se encontrar uma matriz de rotação e o vetor de translação a cada passo. A figura 4.5 ilustra este processo:

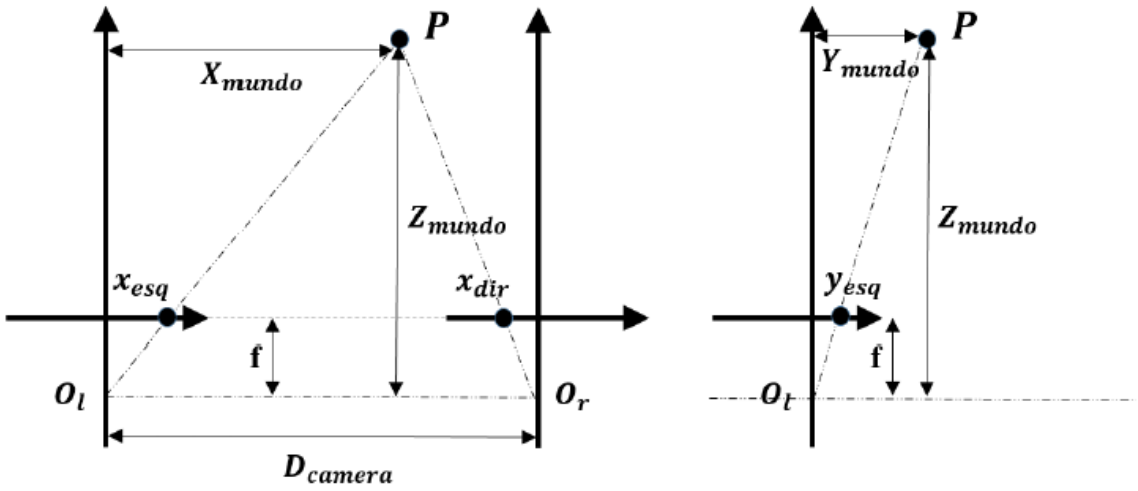
Figura 4.5: Modelo simplificado da composição de movimentação do robô a cada instante.



Fonte: Autor.

A realização dessa composição é possível utilizando câmeras estereoscópicas. A posição do mesmo ponto visto num mesmo instante por duas câmeras é estimada utilizando a geometria epipolar descrita nas próximas equações e com base na Figura 4.6.

Figura 4.6: Geometria perspectiva de câmeras estereoscópicas.



Fonte: Autor.

$$X_m = \frac{x_{esq}}{x_{esq} - x_{dir}} D_{cameras} \tag{4.10}$$

$$Z_m = \frac{X_m}{x_{esq}} f \tag{4.11}$$

$$Y_m = \frac{Z_m}{f} y_{esq} \tag{4.12}$$

Onde $D_{cameras}$ é a distância entre as câmeras, x_{esq} e y_{esq} as coordenadas de câmera do ponto visto pela câmera da esquerda e x_{dir} pela câmera da direita.

Para estimação das matrizes de rotação R_k e do vetor de translação t_k do robô, utiliza-se a seguinte relação:

$$\begin{bmatrix} R_k & t_k \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} P_{w1}(k) & P_{w2}(k) & \dots & P_{wn}(k) \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} P_{w1}(k+1) & P_{w2}(k+1) & \dots & P_{wn}(k+1) \\ 1 & 1 & \dots & 1 \end{bmatrix}^{+} \tag{4.13}$$

Onde $P_{wi}(k), i = 1,2 \dots n$ é o vetor de coordenadas globais de um ponto característico no instante de tempo "k" e + é o operador pseudo-inversa.

O mapa completo de trajetórias é então calculado utilizando a equação 4.14

$$X(k) = X(k-1) + \left(\prod_{i=1}^{k-1} R_i \right) t_k \tag{4.14}$$

Onde $X(k)$ é o vetor de coordenadas globais que é paralelo ao sistema de coordenadas do robô no instante $k = 0$. A matrix R_i e o vetor t_k representam as estimativas de rotação e translação do robô através da captura de duas imagens subsequentes.

Evidentemente, tal estimativa não é perfeita devido aos erros presentes na correlação dos pontos encontrados na imagem e acumulação de erros nos cálculos realizados. O próximo passo é aplicar um filtro para melhoria dos resultados obtidos. A próxima seção descreve o uso de redes neurais neste processo.

4.3 Treinamento e Aplicação da Rede Neural

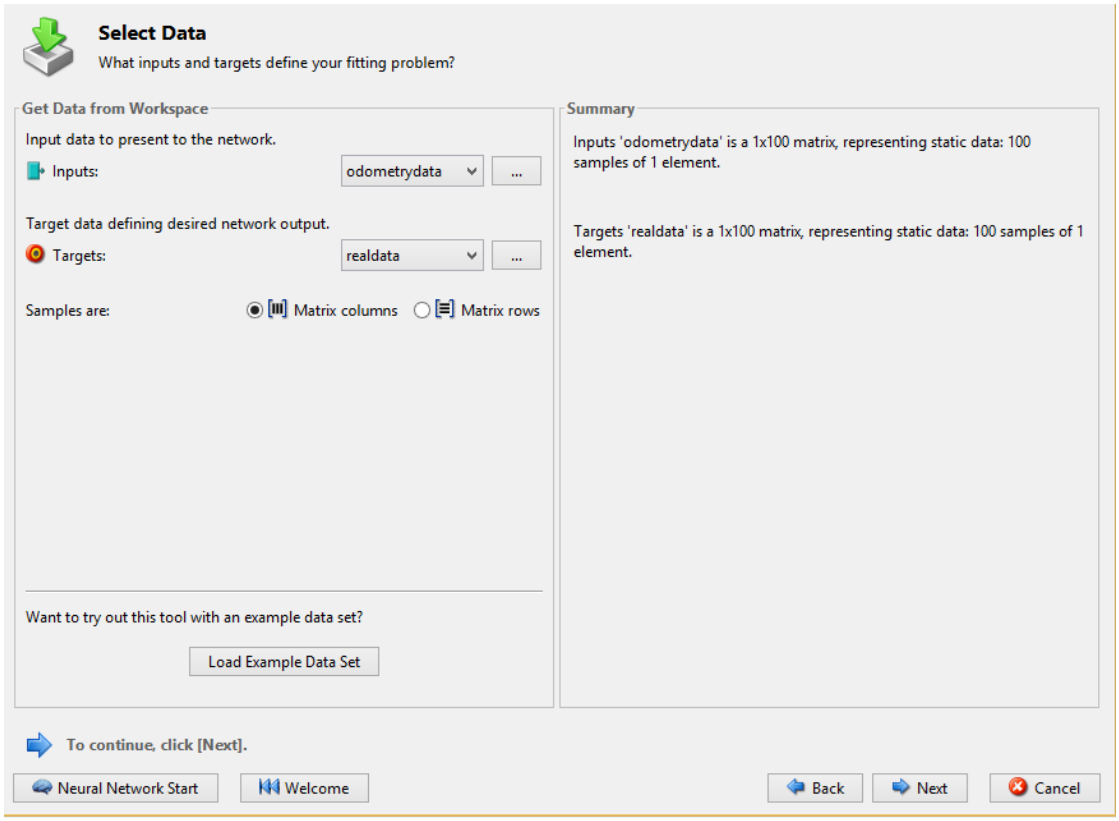
A regularização de trajetórias estimadas por odometria visual é um passo importante para a aplicação no controle de trajetória. Em um projeto de um sistema de controle clássico de malha fechada, o sensor de uma determinada variável é utilizado para emitir dados que são utilizados no cálculo do erro (valor esperado – valor medido) e então o controlador utiliza essa informação para corrigir a dinâmica do processo de maneira adequada. É evidente que quanto maior a imprecisão da medida do sensor, maior a incerteza no cálculo do erro – fazendo com que o controlador atue de forma inadequada.

Nesse caso, a rede neural age como um filtro que tentará, através de técnicas de treinamento, prever o erro da estimativa de odometria visual e então regularizar o resultado de forma a tornar a medição mais próxima do valor real.

O desenvolvimento de uma rede neural pode ser um problema muito complexo. Para isto, existem pacotes prontos que facilitam sua aplicação. Neste projeto, utilizou-se a *toolbox* do Matlab – *nftool* (*Neural Fitting Tool)*. As Figuras de 4.7 a 4.10 ilustram o processo de criação e treinamento da rede.

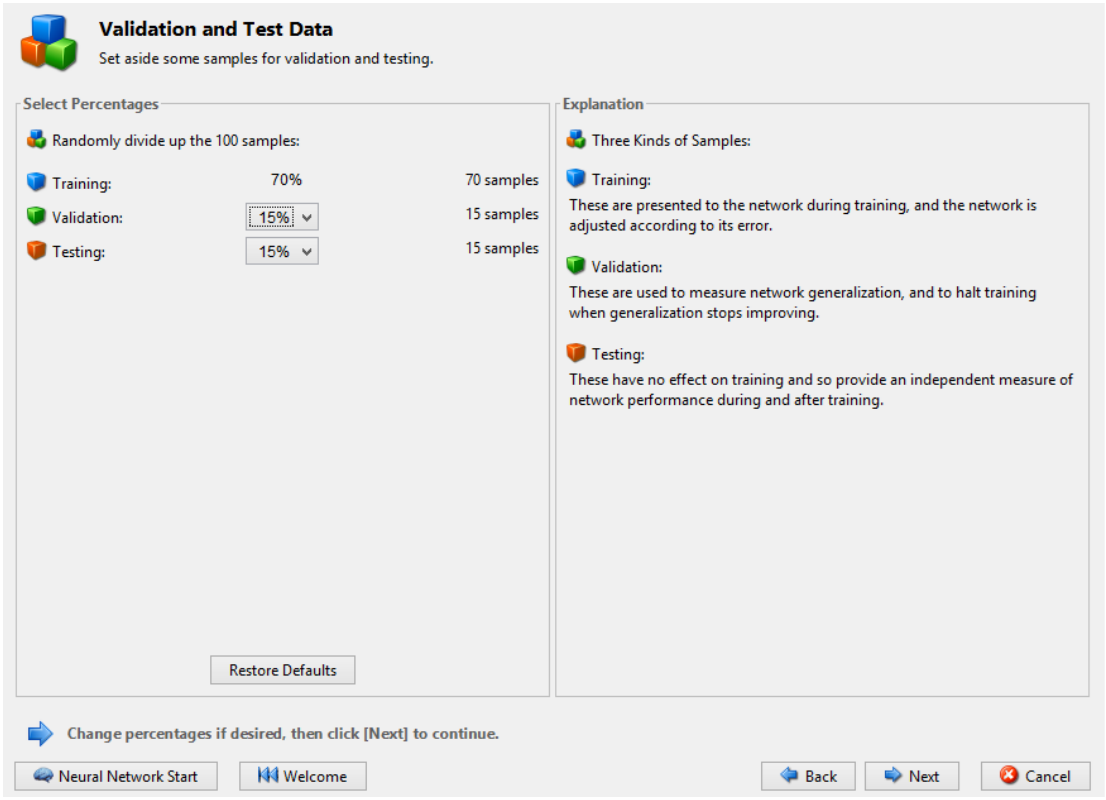
A etapa de seleção de dados (Figura 4.7) é onde são definidas as entradas utilizadas no treinamento da rede e exatamente a saída (*target*) desejada. Neste caso, a entrada é a estimativa do deslocamento realizado pela odometria visual e a saída é a movimentação real (conhecida) do robô.

Figura 4.7: Seleção dos dados de entrada e *targets* para o treinamento.



Fonte: Autor.

Figura 4.8: Etapa de validação e teste dos dados.



Fonte: Autor.

Na validação e teste dos dados (Figura 4.8), são definidas as porcentagens dos dados inseridos na fase de seleção que serão usados para treinamento, validação e teste. Respectivamente, 70%, 15% e 15%. Onde, o primeiro varia de 30% a 70% e os demais de 5% a 35%.

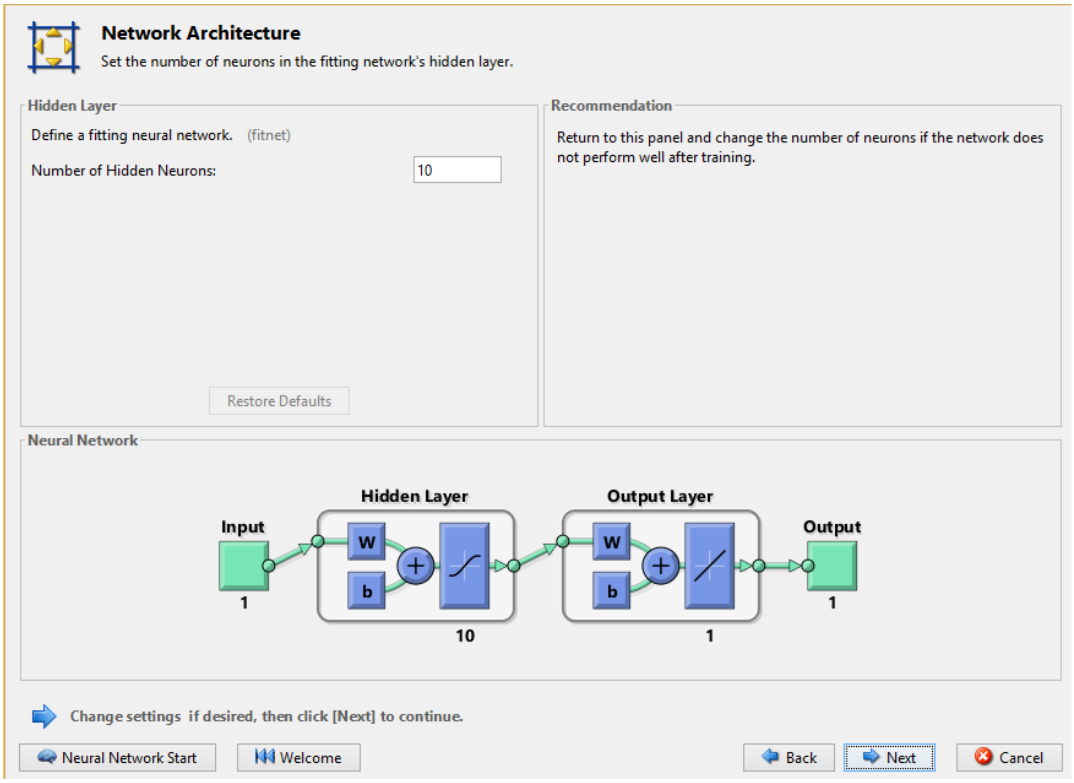
Os dados de treinamento são utilizados no aprendizado e a rede neural é ajustada de acordo com o erro computado.

A validação é utilizada para medir a generalização da rede e é um parâmetro que finaliza o treinamento quando o grau de generalização atinge seu máximo.

O teste não influencia no treinamento. É utilizado para fornecer uma medida independente da performance da rede durante e após o treinamento.

Na etapa da arquitetura da rede (Figura 4.9), deve-se apenas selecionar o número de neurônios. Tal quantidade depende da complexidade do problema e pode ser otimizada empiricamente.

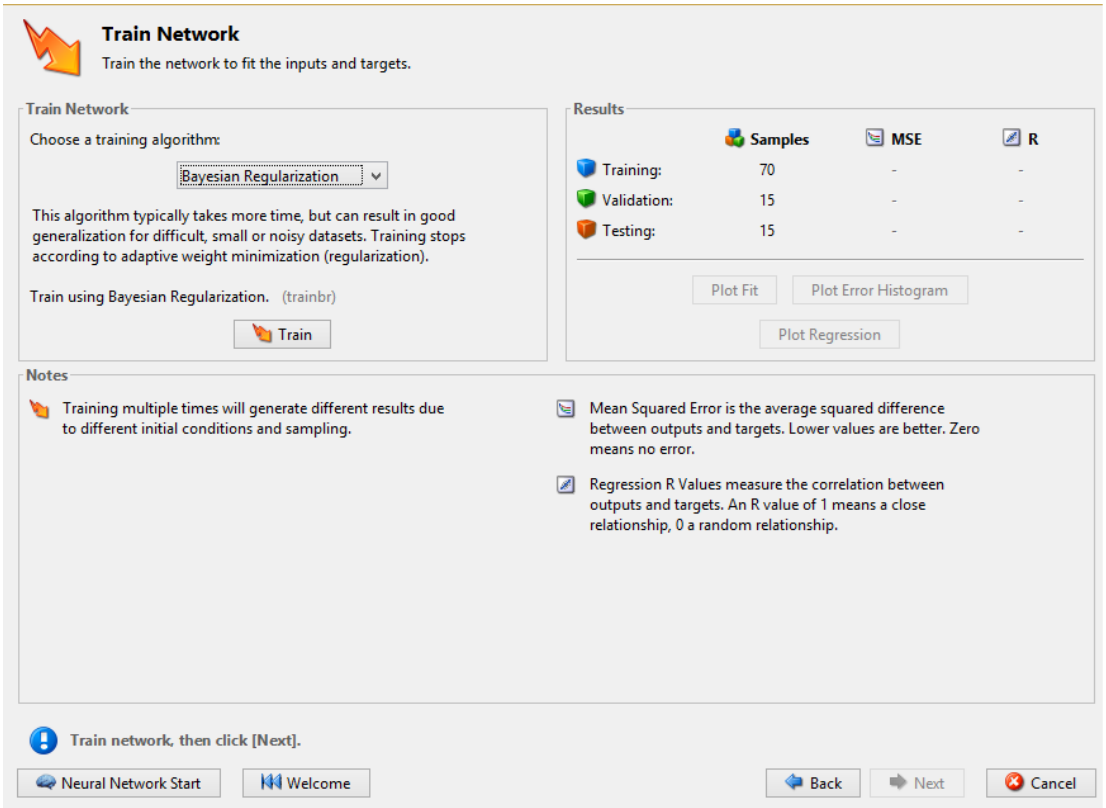
Figura 4.9: Seleção dos dados de entrada e *targets* para o treinamento.



Fonte: Autor.

Finalmente, a etapa de treinamento (Figura 4.10). Nesta, deve-se selecionar o algoritmo de treinamento, dentre os disponíveis na ferramenta estão: *Levenberg-Marquardt, Bayesian Regularization e Scaled Conjugate Gradient*.

Figura 4.10: Etapa de treinamento da Rede Neural.



Fonte: Autor.

Resumidamente, aplicação de rede neural pode ser descrita em dois passos:

1º) Treinamento (Entradas e *Targets* Conhecidos) (*offline*)

- Entrada: X', Y', PSI' (Posições cartesianas e orientação estimadas pela O.V.);
- Target: X,Y,PSI reais (Posições cartesianas e orientações conhecidas);
- Utilização de trajetórias contendo caminhos diversos para treinamento da rede neural (retas horizontais e verticais, diagonais e circulares).

2º) Aplicação da Rede (*online*)

- Depois da rede ser treinada, uma variável do tipo rede neural é criada e então pode ser utilizada como uma função do tipo:

[Saída] = nome_da_rede(Entrada)

5 ANÁLISE EXPERIMENTAL

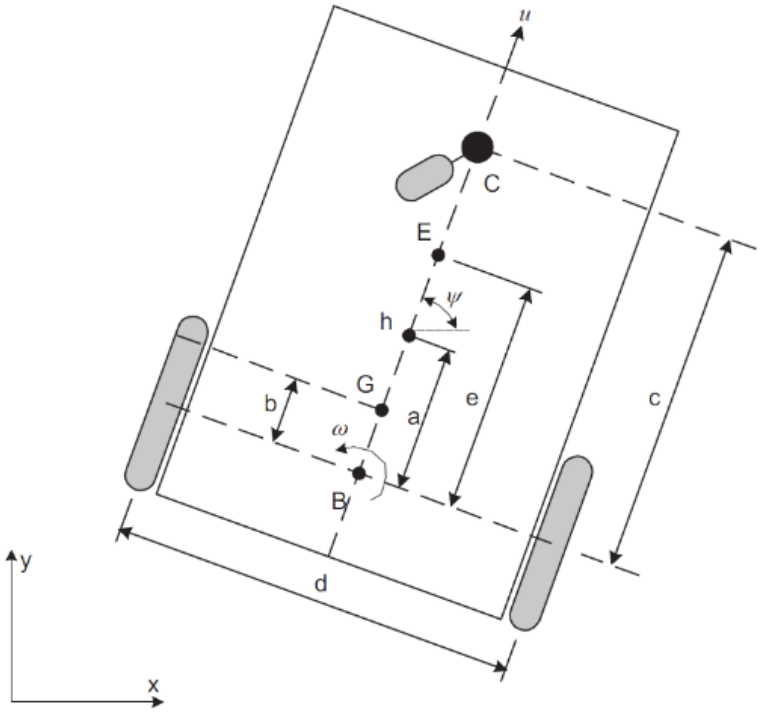
Neste capítulo, será descrito o experimento realizado para a análise da proposta de regularização de trajetórias estimadas por O.V. descritas no capítulo anterior.

Para questões de prova de conceito, toda a análise será feita computacionalmente. Desta forma, é possível realizar diversas simulações mais rapidamente e, com isso, fazer os ajustes necessários para a melhoria do desempenho da técnica ou verificar sua viabilidade.

Primeiramente, como o esperado é aplicar as técnicas de odometria visual em robôs terrestres, precisa-se definir um meio de obter uma trajetória virtual que correspondesse com a movimentação de um robô em um ambiente real.

O trabalho desenvolvido em Bastos (2015) utiliza a dinâmica de um robô diferencial aplicado na ferramenta Simulink do Matlab (Figura 5.1) para navegar em um ambiente virtual (Figura 5.2) desenvolvido no *software* Blender.

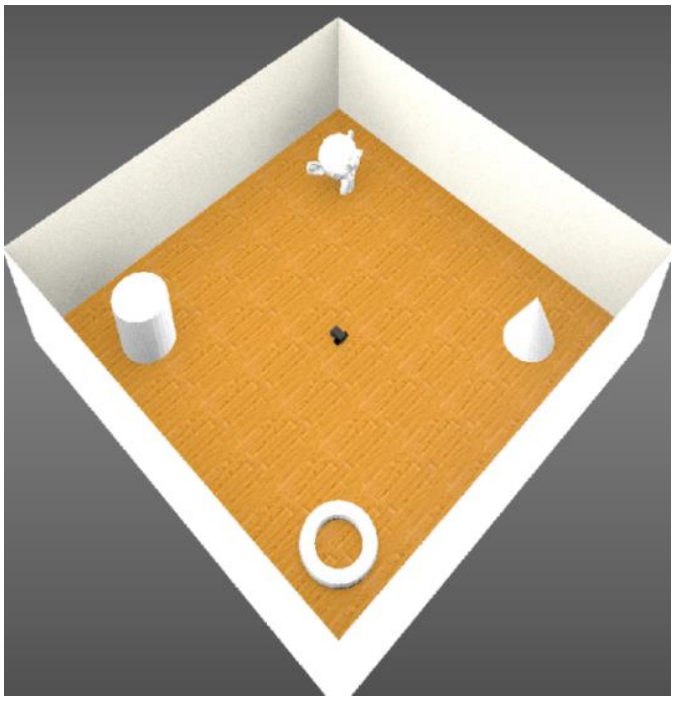
Figura 5.1: Modelo do robô terrestre do tipo diferencial.



Fonte: Bastos (2015)

O robô diferencial possui duas rodas traseiras independentes e uma roda fixa centralizada na parte frontal. Isto faz com que o robô possa girar em torno de seu próprio eixo.

Figura 5.2: Ambiente virtual de uma sala fechada com chão e paredes texturizadas.



Fonte: Bastos (2015).

Esse ambiente de uma sala texturizada é ideal para aplicar técnicas de odometria visual. Para isto, é necessário capturar imagens à medida que o robô se movimenta. Convenientemente, o *software* Blender possui câmeras digitais que podem ser facilmente implementadas – basta monta-las na estrutura do robô.

As câmeras são posicionadas na parte frontal do robô viradas para baixo, isto é, capturando imagens do chão do ambiente. Essa escolha foi feita pelo simples fato de que, considerando que o ambiente é plano, posicionar as câmeras para baixo traz o benefício de que os pontos identificados na parte do processamento sempre terão a mesma profundidade – facilitando e melhorando a estimativa pela odometria visual. A Figura 5.3 ilustra um exemplo de uma imagem capturada no ambiente virtual. A resolução da imagem é de 720x580 *pixels*.

Com a trajetória definida, o robô então se movimenta no ambiente virtual e, em determinados passos, as câmeras capturam fotos do chão ao longo da trajetória. Após a finalização da movimentação do robô, um conjunto de imagens é convertido em um vídeo com a mesma resolução das imagens e com taxa de 30 *frames* por segundo. O vídeo é então carregado pelo Visual Studio onde um algoritmo, baseado na biblioteca que contém diversas ferramentas de processamento de imagens – OpenCV, analisa o vídeo *frame* a *frame* e identifica pontos de interesse para estimar o deslocamento de cada *pixel* selecionado. A Figura 5.4 ilustra um exemplo de tal seleção e a Figura 5.5 o programa em funcionamento.

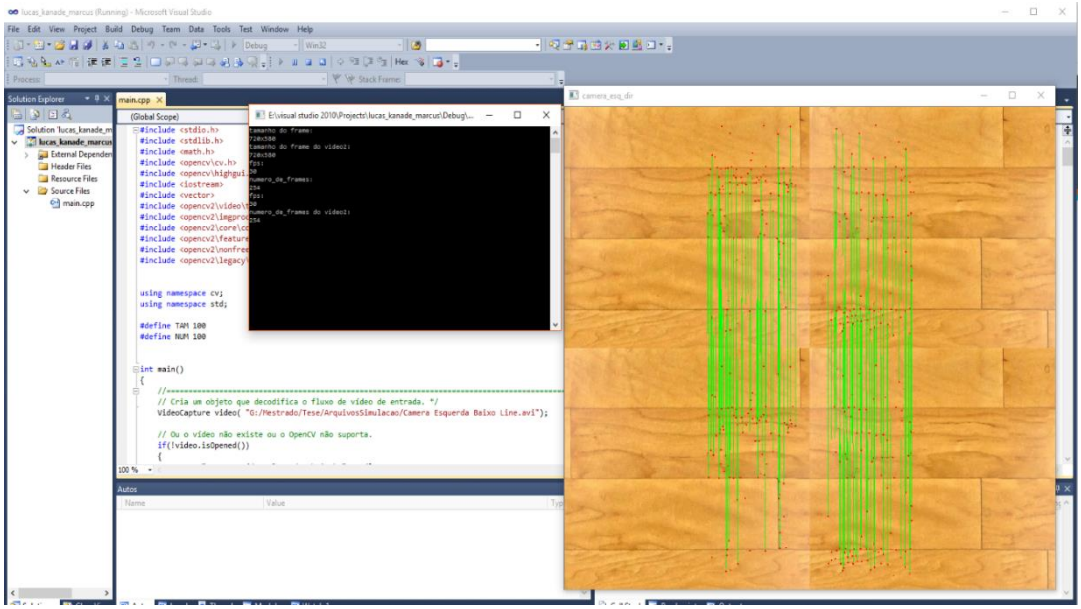
Figura 5.3: Imagem capturada em uma câmera virtual no ambiente do Blender.

Fonte: Autor

Figura 5.4: Identificação de pontos nas imagens da esquerda e direita das câmeras virtuais.

Fonte: Autor

Figura 5-5: Funcionamento do processamento das imagens aplicado no Visual Studio.



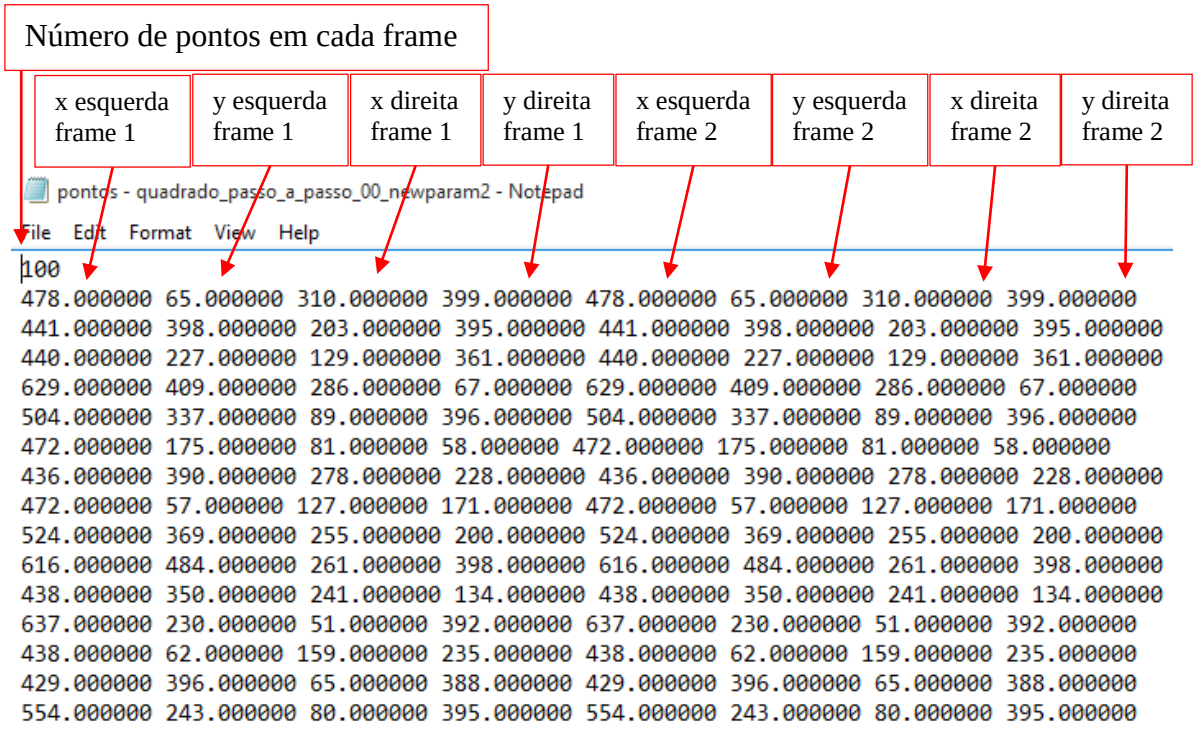
Fonte: Autor

Os pontos são salvos em um arquivo de texto representando as coordenadas de cada ponto em coordenadas de *pixel* (x, y) da imagem da esquerda, direita e os mesmos pontos rastreados no frame subsequente. A Figura 5.5 ilustra tal organização.

O arquivo de texto contendo as informações dos pontos é então lido por um *script* no Matlab, onde toda a álgebra para conversão das coordenadas de *pixel* para coordenadas de mundo e composição da movimentação do robô é realizada.

A princípio, é esperado que a estimativa realizada pela odometria visual forneça um resultado com uma precisão não adequada para a utilização em um controle de trajetória. Tendo isto em consideração, é necessária a aplicação de algum filtro para reduzir o erro gerado pela estimativa, neste caso uma rede neural.

Figura 5.5: Estrutura do arquivo de texto contendo os pontos identificados nas imagens capturadas.

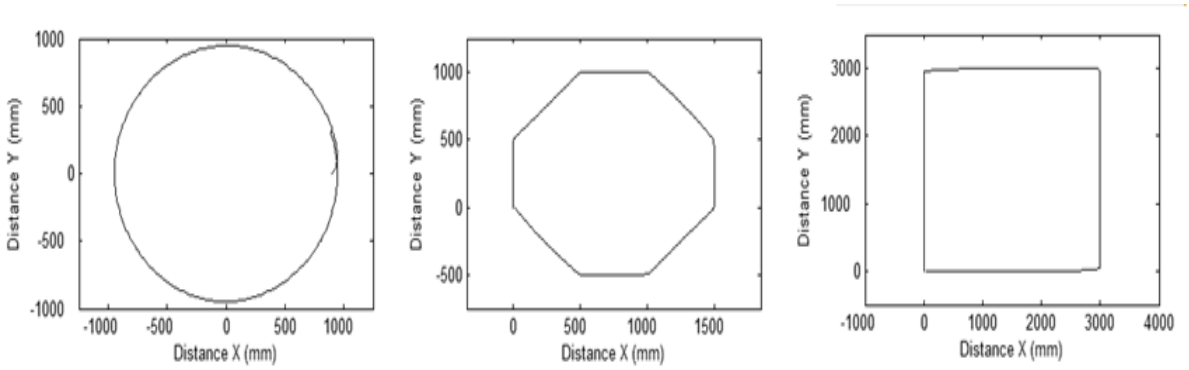


Fonte: Autor.

O treinamento da rede neural é feito utilizando as trajetórias da Figura 5.6. A escolha dessas trajetórias se dá pelo motivo de dar a capacidade para a rede neural aprender a corrigir os erros em tipos de caminhos diferentes: retas horizontais, verticais e diagonais e curvas.

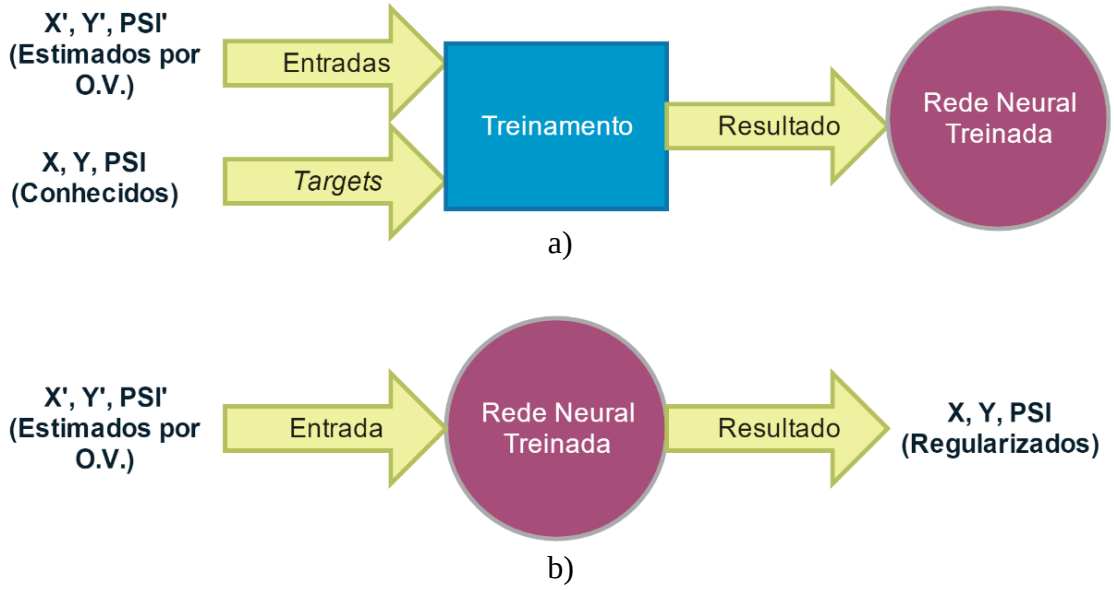
Com isso, espera-se que qualquer trajetória realizada pelo robô, composta pelos caminhos do treino, apresente um bom resultado de estimativa. A Figura 5.7 ilustra um esquema do treinamento e aplicação da rede neural.

Figura 5.6: Trajetórias circular, octogonal e quadrada aplicadas no treinamento da rede neural.



Fonte: Autor.

Figura 5.7 Fluxogramas da etapa de treinamento (a) e aplicação da rede neural (b).



Fonte: Autor.

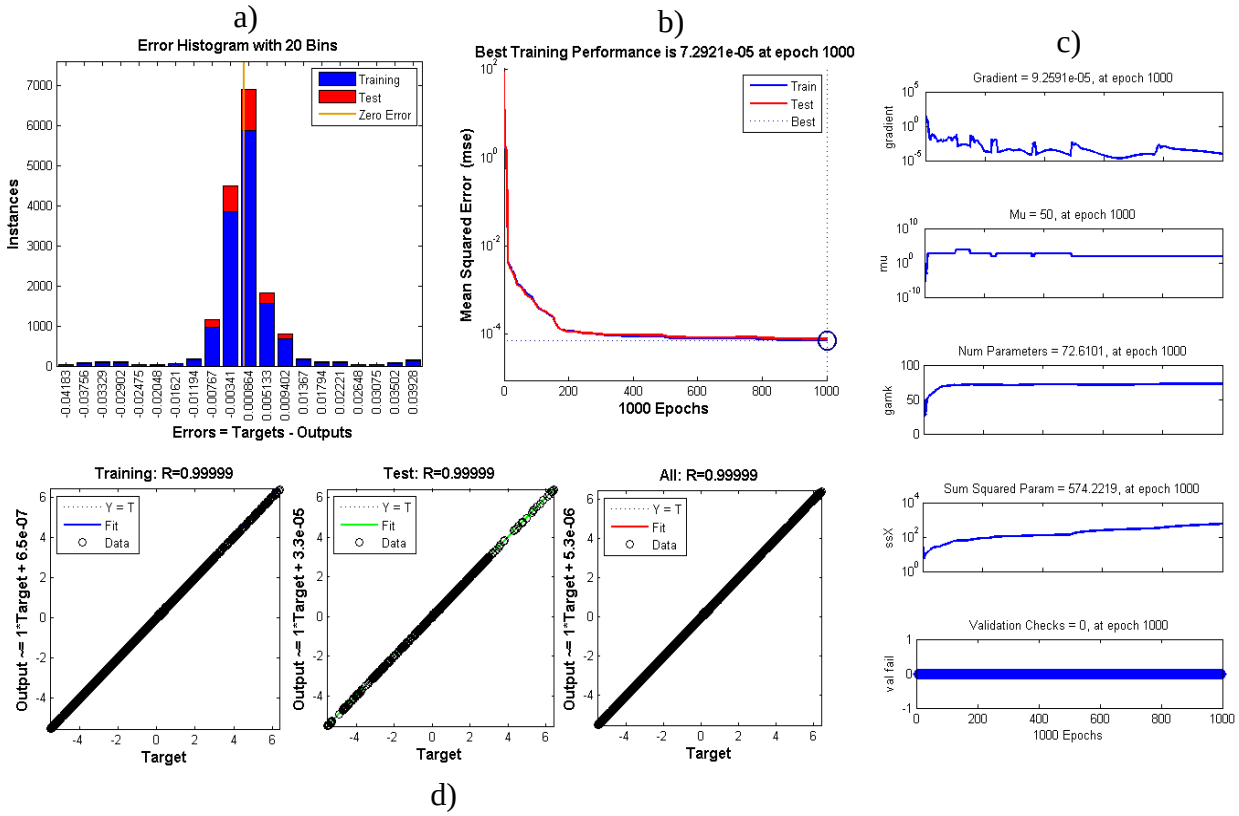
Com a rede neural treinada, pode-se aplicar uma trajetória ao robô que seja composta das trajetórias de treinamento. A expectativa é que haja uma melhora significativa da estimativa.

6 RESULTADOS

Neste capítulo, são apresentados os resultados do treinamento da rede neural, a estimativa por odometria visual das trajetórias aplicadas, a regularização realizada pela rede neural treinada e o trabalho científico gerado com este estudo.

Primeiramente, sobre o treinamento da rede neural, a Figura 6.1 apresenta os resultados obtidos durante o treinamento da rede realizado com as trajetórias descritas no capítulo anterior.

Figura 6.1: Resultado da performance da rede neural após o treinamento: Histograma de erro (a), Erro Médio Quadrático/Época (b), Parâmetros da rede neural (c) e Saída/*Target* dos dados de treinamento e teste (d).



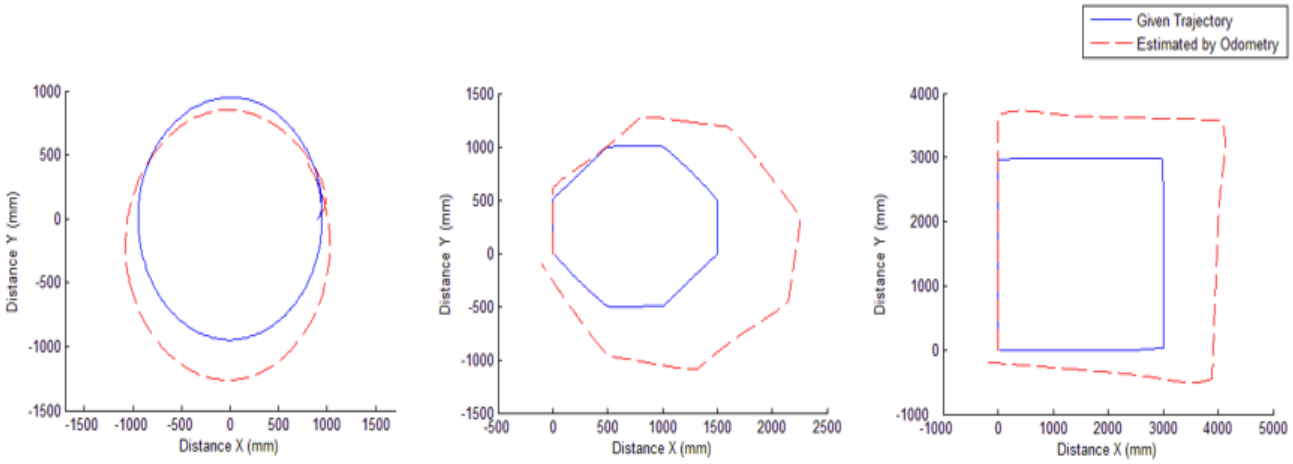
Fonte: Autor.

O histograma de erro com 20 *bins* apresentou um resultado desejado. Uma distribuição gaussiana, onde a maior concentração está no meio (menor erro).

A performance de treinamento da rede visto na Figura 6-1b atinge um resultado bom em aproximadamente 200 épocas e o melhor resultado em 1000 épocas. O rápido declínio do erro indica que a rede está configurada corretamente.

A Figura 6.2 mostra o resultado da estimativa de odometria visual apenas nas trajetórias utilizadas no treinamento da rede neural.

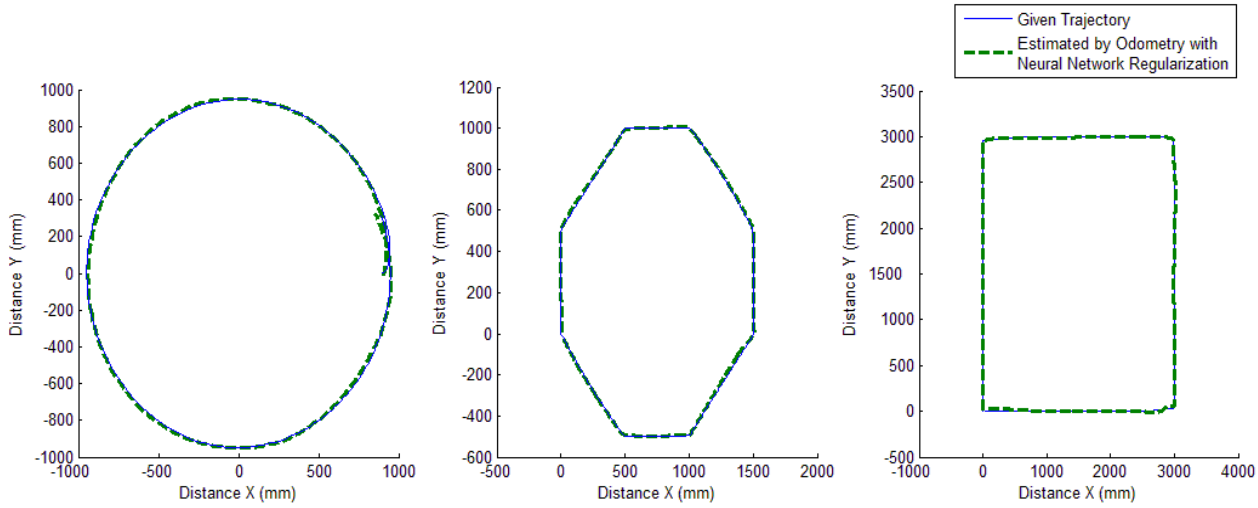
Figura 6.2: Resultado da estimativa das trajetórias circular, octogonal e quadrada por odometria visual.



Fonte: Autor

Percebe-se um erro em relação à escala e rotação. O próximo passo é testar a rede neural treinada. A Figura 6.3 ilustra a aplicação das trajetórias utilizadas para o treinamento na rede neural treinada.

Figura 6.3: Resultado da aplicação das trajetórias circular, octogonal e quadrada na rede neural treinada.



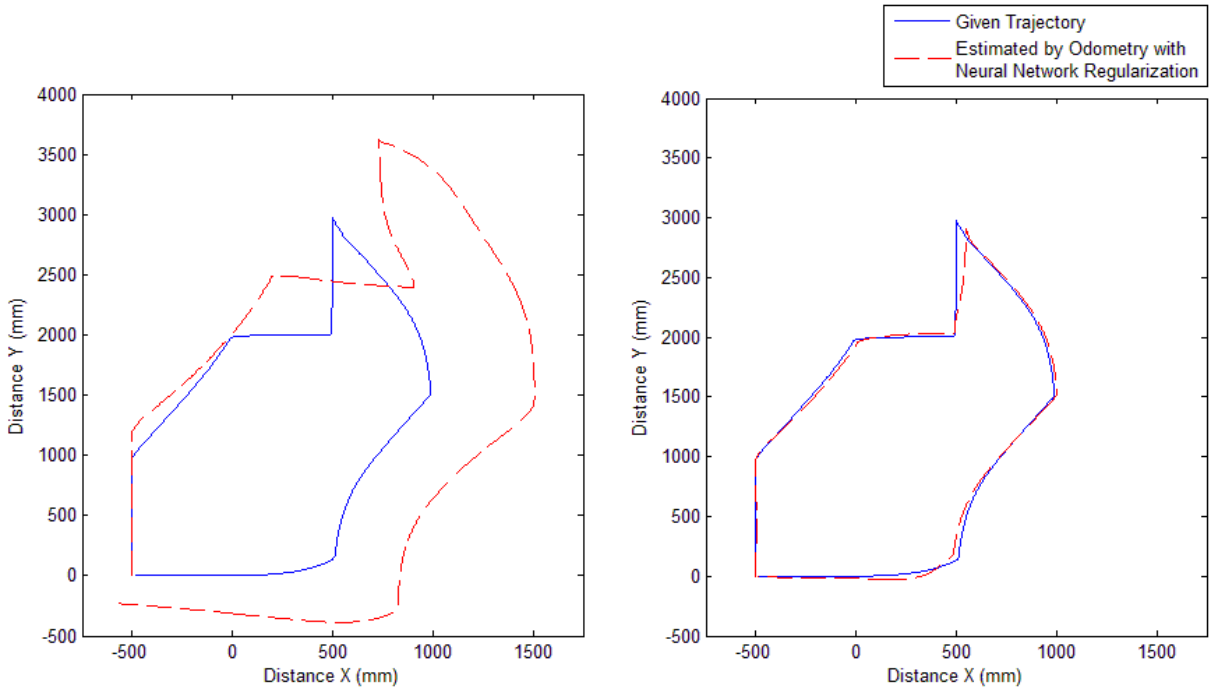
Fonte: Autor

Como esperado, a rede neural conseguiu regularizar as trajetórias com um erro mínimo. Isto só comprova que a rede foi treinada adequadamente, já que aplicar a rede neural treinada

nas trajetórias de treinamento não fornece informações da performance da rede em caminhos diversos.

Para testar o desempenho da rede neural, define-se uma trajetória que deve ser composta por parte das trajetórias de treinamento: retas horizontais, verticais, diagonais e curvas circulares. A Figura 6.4 ilustra tal trajetória composta estimada apenas por odometria visual e regularizada pela rede neural.

Figura 6.4: Trajetória composta estimada apenas por odometria visual (esquerda) e regularizada por rede neural (direita).



Fonte: Autor.

O resultado estimado apenas por odometria visual, assim como nas trajetórias de treinamento, apresentam um erro de escala e rotação. Com a aplicação da rede neural, a resposta é melhorada de maneira significante, um erro máximo aproximadamente de 100mm em “x” e 50mm em “y”. As regiões de maior erro são principalmente nas curvas, onde a velocidade angular do robô dificulta a correlação de pontos. Uma solução para tal problema é diminuir a velocidade do robô antes das curvas. A tabela 6.1 mostra os resultados de forma quantitativa.

Verifica-se que, em todos os casos, há uma diminuição significante no erro médio quadrático, com mais relevância na trajetória composta, já que esta não foi utilizada no treinamento da rede e sim para validação. A tabela 6.2 ilustra o tempo de execução das etapas.

A execução do algoritmo de processamento de imagens é realizada no *software* Visual Studio, a odometria e a rede neural são aplicadas no Matlab. O computador utilizado tem o sistema operacional Windows 7, processador Intel I7 4790k de 4.0Ghz e placa de vídeo GeForce GTX 660.

Tabela 6.1: Erro médio quadrático das posições com odometria visual e rede neural

	Erro Médio Quadrático			
Trajetória	Posição X (Real vs. O.V.)	Posição Y (Real vs. O.V.)	Posição X (Real vs. O.V. + R.N.)	Posição Y (Real vs. O.V. + R.N)
Quadrado	0.4285	0.2027	$1.86 \cdot 10^{-5}$	$1.13 \cdot 10^{-5}$
Octógono	199.1667	43.7371	$0.67 \cdot 10^{-3}$	$0.19 \cdot 10^{-3}$
Círculo	0.0272	8.2991	$0.66 \cdot 10^{-3}$	$0.07 \cdot 10^{-5}$
Composta	361.0754	26.8454	0.4853	0.0216

Fonte: Autor

Tabela 6.2: Tempo de execução da odometria visual, treinamento e aplicação da rede.

	Tempo de Execução				
Trajetória	Comprimento do Percurso	Processamento das Imagens	Odometria Visual	Treinamento da Rede	O.V. + Aplicação da Rede
Quadrado	12 metros	$3m44s$	$1m34s$	$2m05s$	$1m42s$
Octógono	4.83 metros	$5m57s$	$2m46s$	$2m05s$	$2m51s$
Círculo	6.28 metros	$4m33s$	$1m53s$	$2m05s$	$1m58s$
Composta	7.78 metros	$8m45s$	$3m26s$	-	$3m32s$

Fonte: Autor

Claramente, o maior tempo de execução é no processamento de imagens, isto ocorre devido à resolução relativamente grande das imagens e também da quantidade de passos do robô. O tempo médio de atraso no processo em cada passo do programa com a rede trainada é de aproximadamente 1.18 segundos. Este tempo pode ser reduzido com a melhoria dos algoritmos de processamento e estimativa e também com a unificação de todo o processo em um único *software*.

O treinamento da rede pode ser relativamente demorado, dependendo da quantidade de dados inseridos. A grande vantagem é que, após o treinamento, a rede adiciona pouco tempo de processamento em sua aplicação e melhora os resultados de maneira relevante.

O estudo apresentado neste trabalho gerou, uma apresentação no congresso internacional IPIN-2015 onde o artigo será publicado nos anais do evento. O artigo gerado é apresentado no apêndice A.

7 CONCLUSÕES E TRABALHOS FUTUROS

A navegação autônoma por odometria visual traz muitas vantagens em relação aos métodos tradicionais. A não dependência de um sistema de navegação por satélite possibilita a navegação em ambientes isolados, subterrâneos ou sem cobertura de sinal. Técnicas atuais que utilizam um sistema de *lasers* fornecem uma grande quantidade de informações complexas para o processamento e dependem do índice de reflexão dos materiais. A utilização de imagens traz uma grande quantidade de informações que podem ser extraídas. Dentre elas: separação, identificação e contagem de objetos, segmentação de áreas. É possível também introduzir o conceito de profundidade (reconstrução 3-D) descrita na proposta deste trabalho utilizando imagens estereoscópicas e conhecendo os parâmetros intrínsecos da câmera e a distância entre ambas.

Em contrapartida, o processamento de imagens é altamente dependente das condições de iluminação do ambiente, bem como questões de estabilidade (vibração). Este projeto considera a navegação em um ambiente plano e com iluminação distribuída regularmente (ambiente estruturado).

Utilizando apenas a odometria visual com as técnicas descritas neste projeto, é possível prever a forma e o tamanho da trajetória do robô com certo erro na escala e rotação. Apesar de ser um resultado relativamente bom, a estimativa pura não poderia ser utilizada adequadamente para controle de uma trajetória.

A aplicação da rede neural para regularização do erro presente na estimativa de odometria visual se mostrou muito eficiente. Diferente de outros filtros, a rede neural adiciona pouco tempo de processamento em relação ao resultado trazido. Por outro lado, a rede neural pode ter um elevado tempo de treinamento e geralmente é sensível às mudanças na aplicação não consideradas no treinamento. Uma proposta para resolver tal questão é criar diversos ambientes virtuais com características diferentes, treinar uma rede neural para cada um e no final ter um conjunto de redes neurais que podem ser selecionadas para cada situação. Este método pode ser inclusive dinâmico – quando o robô passa de um determinado tipo de terreno para outro, é selecionado um tipo de rede mais adequada para a situação.

Sabe-se que, na aplicação das técnicas descritas neste trabalho em uma situação real, podem ocorrer grandes disparidades na estimativa de odometria visual devido às questões de

iluminação e vibração. Isto sugere um estudo sequencial que aplique técnicas de normalização da imagem a fim de minimizar tais efeitos indesejados.

Outra questão é a elevação do ambiente. Para ambientes internos isto não é tão alarmante, contudo, para uma navegação em ambientes externos, isto deve ser considerado no treinamento da rede. Uma sugestão para solucionar este problema seria introduzir na rede neural mais uma variável. Além de $(x', y' e\ psi')$, ter-se-ia uma variável h (elevação) que poderia ser medida tanto por processamento de imagens quanto por um sensor inercial, por exemplo.

A execução deste projeto só foi possível pela integração de três programas: Matlab, Blender e Visual Studio. Utilizar três *softwares* diferentes traz uma grande quantidade de funcionalidades diferentes. Todavia, tal comunicação pode gerar problemas de conflito e acaba reduzindo muito a eficiência no tempo de processamento do processo. O próximo passo é converter todo o processamento para um único aplicativo, preferencialmente programado em uma linguagem que possa ser embarcada como: *Python* ou *Java*. Isto definitivamente irá reduzir o tempo de processamento e praticamente eliminar problemas de compatibilidade, além de poder ser aplicado em um dispositivo móvel: um celular, *tablet* ou uma placa processadora.

Ainda que o fluxo óptico tenha resolvido o problema proposto, existem outras técnicas de processamento de odometria visual como Sift/Surf (Suaib, 2014) e ORB (Rublee, 2011) que precisam ser avaliadas. A rede neural também pode ser substituída por um filtro de Kalman (Chen, 2012) ou um descritor do tipo SVM-KNN (Zhang, 2006). Uma possível tese de doutorado seria um estudo aprofundado das técnicas mencionadas realizando testes comparativos e então propor qual seria a melhor configuração, em termos quantitativos e qualitativos, para uma navegação autônoma visual com melhor estimativa de posição.

Por fim, com o estudo realizado, verificou-se que o tema de navegação autônoma visual é bastante discutido na atualidade e que ainda não possui um caminho nem uma solução ótima para o problema. Por ser uma questão abrangente, este trabalho levante diversos aspectos que podem ser temas de dissertações de mestrado ou até teses de doutorado. Acredita-se que este tipo de estudo tem uma grande relevância militar, principalmente na exploração planetária.

REFERÊNCIAS

AGRAWAL, Motilal; KONOLIGE, Kurt. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In: **Pattern Recognition, 2006. ICPR 2006. 18th International Conference on**. IEEE, 2006. p. 1063-1068.

BASTOS, Vinicius Benites. **Desenvolvimento e Integração de Ambiente 3D de Simulação para Algoritmos de Navegação por Imagem.** 2015. 117 f. Dissertação (Mestrado) - Curso de Engenharia Mecânica, Unicamp, Campinas, 2015.

BURGUERA, Antoni; GONZÁLEZ, Yolanda; OLIVER, Gabriel. RANSAC Based Data Association for Underwater Visual SLAM. In: **ROBOT2013: First Iberian Robotics Conference**. Springer International Publishing, 2014. p. 3-16.

CANNY, John. A computational approach to edge detection. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, n. 6, p. 679-698, 1986.

CHEN, S. Y. Kalman filter for robot vision: a survey. **Industrial Electronics, IEEE Transactions on**, v. 59, n. 11, p. 4409-4420, 2012.

COLLINS, Robert. **Correspondence Matching.** Penn State: Collins, 2012. 33 slides, color. Disponível em: <http://www.cse.psu.edu/~rtc12/CSE486/lecture07.pdf>. Acesso em: 14 abr. 2015.

DOUXCHAMPS, Damien. **A small list of IMU / INS / INU.** 2015. Disponível em: <http://damien.douxchamps.net/research/imu/>. Acesso em: 14 abr. 2015.

DUDA, Richard O. et al. **Pattern classification and scene analysis**. New York: Wiley, 1973.
FÖRSTNER, Wolfgang. A feature based correspondence algorithm for image matching. **International Archives of Photogrammetry and Remote Sensing**, v. 26, n. 3, p. 150-166, 1986.

FU, Changhong et al. Monocular visual-inertial SLAM-Based collision avoidance strategy for fail-safe UAV using fuzzy logic controllers. **Journal of Intelligent & Robotic Systems**, v. 73, n. 1-4, p. 513-533, 2014.

HARRIS, Chris; STEPHENS, Mike. A combined corner and edge detector. In: **Alvey vision conference**. 1988. p. 50.

KAO, Wei-Wen; HUY, Bui Quang. Indoor navigation with smartphone-based visual SLAM and Bluetooth-connected wheel-robot. In: **Automatic Control Conference (CACS), 2013 CACS International**. IEEE, 2013. p. 395-400.

KONOLIGE, Kurt; AGRAWAL, Motilal; SOLA, Joan. Large-scale visual odometry for rough terrain. In: **Robotics Research**. Springer Berlin Heidelberg, 2011. p. 201-212.

LATEGAHN, Henning; GEIGER, Andreas; KITT, Bernd. Visual SLAM for autonomous ground vehicles. In: **Robotics and Automation (ICRA), 2011 IEEE International Conference on**. IEEE, 2011. p. 1732-1737.

LEBLANC, Simon. **Dynamics of group hunting and collective evasion in schooling fish.** 2014. Disponível em: <http://www.princeton.edu/~sleblanc/en/research.html>. Acesso em: 14 abr. 2015.

LIU, Jiamin; WHITE, Jacob M.; SUMMERS, Ronald M. Automated detection of blob structures by hessian analysis and object scale. In: **Image Processing (ICIP), 2010 17th IEEE International Conference on**. IEEE, 2010. p. 841-844.

MACKAY, David JC. Bayesian interpolation. **Neural computation**, v. 4, n. 3, p. 415-447, 1992.

MEDINA-CARNICER, Rafael et al. A novel method to look for the hysteresis thresholds for the Canny edge detector. **Pattern Recognition**, v. 44, n. 6, p. 1201-1211, 2011.

MØLLER, Martin Fodslette. A scaled conjugate gradient algorithm for fast supervised learning. **Neural networks**, v. 6, n. 4, p. 525-533, 1993.

MORAVEC, Hans P. TOWARDS AUTOMATIC VISUAL BBSTACLE AVOIDANCE. In: **International Conference on Artificial Intelligence (5th: 1977: Massachusetts Institute of Technology)**. 1977.

MORÉ, Jorge J. The Levenberg-Marquardt algorithm: implementation and theory. In: **Numerical analysis**. Springer Berlin Heidelberg, 1978. p. 105-116.
NASA. **Mars Curiosity.** Disponível em: <http://www.nasa.gov/mission_pages/msl/images/index.html>. Acesso em: 14 abr. 2015.

NOWICKI, Michal; SKRZYPCZYNSKI, Piotr. Performance comparison of point feature detectors and descriptors for visual navigation on Android platform. In: **Wireless**

Communications and Mobile Computing Conference (IWCMC), 2014 International. IEEE, 2014. p. 116-121.

PARKER, Jim R. **Algorithms for image processing and computer vision**. John Wiley & Sons, 2010.

RUBLEE, Ethan et al. ORB: an efficient alternative to SIFT or SURF. In:**Computer Vision (ICCV), 2011 IEEE International Conference on**. IEEE, 2011. p. 2564-2571.

SHI, Jianbo; TOMASI, Carlo. Good features to track. In: **Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on**. IEEE, 1994. p. 593-600.

SUAIB, Norhayati Mohd et al. Performance evaluation of feature detection and feature matching for stereo visual odometry using SIFT and SURF. In:**Region 10 Symposium, 2014 IEEE**. IEEE, 2014. p. 200-203.

SYSTEMS, Phoenix Aerial. **Comparison of Inertial Measurement Units (IMUs).** 2014. Disponível em: <http://www.phoenix-aerial.com/information/imu-comparison/>. Acesso em: 13 abr. 2015.

WEDEL, Andreas; CREMERS, Daniel. **Stereo scene flow for 3D motion analysis**. Springer Science & Business Media, 2011.

WEISS, Stephan; SCARAMUZZA, Davide; SIEGWART, Roland. Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments. **Journal of Field Robotics**, v. 28, n. 6, p. 854-874, 2011.

ZHANG, Hao et al. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In: **Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on**. IEEE, 2006. p. 2126-2136.

ZHANG, Tao; XU, Xiaosu. A new method of seamless land navigation for GPS/INS integrated system. **Measurement**, v. 45, n. 4, p. 691-701, 2012.

APÊNDICE A – ARTIGO APRESENTADO NO IPIN 2015

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 13-16 October 2015, Banff, Alberta, Canada

vSlam Experiments in a Custom Simulated Environment

Marcus V P Lima, Vinicius B Bastos, Paulo R G Kurka, Darla C Araujo
University of Campinas
Integrated Systems Laboratory
Campinas, Brazil
mvpleng@fem.unicamp.br

***Abstract*— In order to perform vSLAM experiments, a camera capturing information along a certain path is necessary. What if the resources to execute this are not available or a proof of concept is needed before a real application? This paper describes a simple method of combining MATLAB and Blender to perform vSLAM experiments in a custom created environment. A virtual robot and camera are created to navigate through the simulated space to capture textured images and objects that are used for the image processing. The images or video created by Blender can be easily processed by MATLAB as shown in this work. In addition, a simple application is demonstrated in the experiments section.**

Keywords— vSLAM; Simulated Environment; MATLAB; Blender;

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) is a primary task for the navigation of autonomous robots [1]. The use of images to do such localization changes the abbreviation to vSLAM, where the "v" stands for visual. In the past years, a grown interest in Visual Odometry (VO) is observed, as shown in [2] and [3]. The use of cameras as sensors for mobile robot navigation seems to have many advantages in comparison to other alternatives. In first place, cameras are generally cheaper than most of other sensors such as laser, sonar, GPS, etc… in addition, they are low energy consumption devices and offer a bigger flexibility considering the high amount of information that can be extracted from its images. The process of visual odometry can be divided in three steps: detection of feature points in an image (corners and blobs), feature matching (identifying similar features in different images) and estimation of trajectory (using stereoscopic images and epipolar geometric relations). A variety of algorithms can perform reliable detection of features, such as the Shi-Tomasi[4] and Harris[5] corner detectors and the SIFT[6], SURF[7] detectors of bubbles and the Optical Flow technique [8] .

A 3D creation suite can be used to test such techniques in rendered images using high quality textures to give more realistic characteristics – as if the experiments were executed in a real environment. Blender [9] is an open-source 3D suite with those described features plus a Python scripting area to perform custom applications.

This work presents a method of using computational-only tools to perform experiments of vSLAM. The process is divided in three steps: robot dynamics, virtual environment and integration. The first step describes the implementation of the desired robot's dynamics in MATLAB and the use of existing projects for direct application. In the virtual environment section, the creation of the virtual robot and cameras, the texturing method and the necessary scripting to execute the navigation and acquire image data are described. The final step contains explanation of the integration of MATLAB and Blender for usage in vSLAM experiments.

The application of the proposed method is shown in section V. A simple differential robot is programmed to follow certain paths (in according to its dynamics) in a simulated indoor environment and through the optical flow technique, the path is estimated using the digital images captured in Blender

II. ROBOT DYNAMICS

This section describes the implementation of the robot's dynamics to be used in the vSLAM experiment. This is not a required step, but highly recommended to create a more realistic movement of the robot, in according to its behavior in a real application.

The work in [10] presents the creation of a robotics toolbox for MATLAB. This package allows the readily creation and manipulation of datatypes fundamental to robotics such as homogeneous transformations, quaternions and trajectories. This project is required for the application shown in this work.

The creation of a robotic system in Simulink can be exemplified in work [11], showing the parameter identification of real robots for simulation purposes. The result is a Simulink project that uses the robot's real parameters to perform navigation tests. Fig. 1 shows a simple system used for the robot dynamics simulation created with the tools described above.

The robot type chosen in this work is a unicycle-like. De la Cruz and Carelli[12] proposed a dynamic model (shown in Eq. 1) for this kind of robot represented in Fig. 2. The parameters and variables of interest in the model are the linear and angular

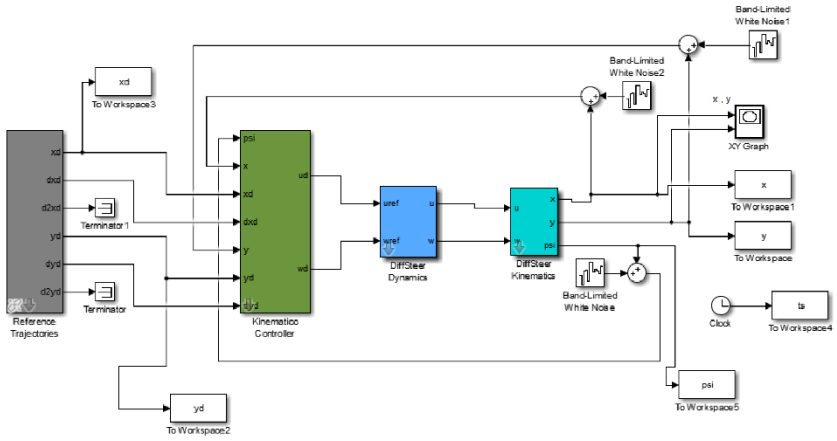


Fig. 1 – Differential robot system in Simulink

velocities of the robot, u and ω, respectively, the robot orientation angle ψ and the distance between wheels, a.

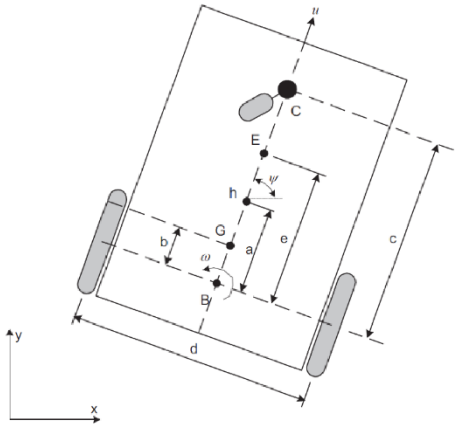


Fig. 2. The unicycle-like mobile robot [11]

The kinematics equation of motion of such a robot is given as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \cos\psi & a\sin\psi \\ 0 & 1 \\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} = \begin{bmatrix} u\cos\psi - a\omega\sin\psi \\ u\cos\psi + a\omega\sin\psi \\ \omega \\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u \\ -\frac{\theta_5}{\theta_2}u\omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix} \quad (1)$$

where:

$$\theta_1 = \left[\frac{R_a}{k_a}(mr^2 + 2I_e) + 2rk_{DT}\right]\frac{1}{2rk_{PT}}[s]$$

$$\theta_2 = \frac{\left[\frac{R_a}{k_a}\left(I_e d^2 + 2r^2(I_z + mb^2)\right) + 2rdk_{DR}\right]}{2rdk_{PR}}[s],$$

$$\theta_3 = \frac{R_a}{k_a}\frac{mbr}{2k_{PT}}[sm/rad^2],$$

$$\theta_4 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{1}{rk_{PT}} + 1\ [1], \qquad \theta_5 = \frac{R_a}{k_a}\frac{mbr}{dk_{PR}}[s/m],$$

$$\theta_6 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{d}{2rk_{PR}} + 1\ [1].$$

In a practical implementation of the robot's model, the kinematics equation must include the desired values of the linear and angular velocities, u_{ref} and ω_{ref}, respectively, a vector of identified parameters θ and uncertainties of the position and velocities δ, yielding:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \cos\psi & a\sin\psi \\ 0 & 1 \\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \\ \delta_u \\ \delta_\omega \end{bmatrix} \quad (2)$$

$$\boldsymbol{\theta} = [\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6]^T \qquad \boldsymbol{\delta} = [\delta_x\ \delta_y\ 0\ \delta_u\ \delta_\omega]^T$$

The input of the system is the desired trajectory and the output is the vector of positions ("x,y") and orientation ("psi") at each iteration defined in the Simulink project. The array of positions is used by Blender to perform the input trajectory. This process is detailed in section IV.

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 13-16 October 2015, Banff, Alberta, Canada

III. VIRTUAL ENVIRONMENT

In this section, the virtual environment created with Blender for the vSLAM applications is detailed. This open-source software is great for photorealistic 3D visualizations, therefore makes a perfect tool to be used with image processing due to the high quality of image details. The literature [13] describes the process of creating such realistic environments.

A sketch of a robot can be easily created using the forms available in the software: plane, cube, circle, sphere, cylinder, cone etc. An example of a robot used in the application described in section V is shown if Fig. 3 The robot shape will

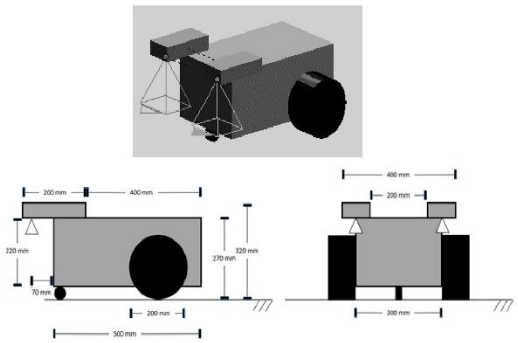


Fig. 3 – Sketch of the differential robot created in Blender

not have any effect, since the dynamics is only presented in the MATLAB model. However, it is possible to add effects such as wind and gravity by a custom Python script in the Blender software.

Scripting in Blender is supported by a large number of built-in and custom-created functions available in Blender's Org. Forum. To move and translate the robot, only 3 lines of code are used:

```
bpy.data.objects['Robot_001'].select = True
bpy.ops.transform.translate(value=(x,y,0))
bpy.ops.transform.rotate(value= r, axis=(0,0,1))
```

The first line is to select the object, then the command "bpy.ops.transform.translate" sends the object to a determined "x,y,z" position. Since in the application in this work is 2D, we set the 'z' to be zero. Also, the rotation of the robot corresponding to is orientation is on the vertical axis ('z' in Blender) and the 'x' and 'y' rotation are zero.

The camera can be easily added as it is already built in the software. Characteristics such as resolution, focus, sensor size, frame rate and aspect ratio are editable. Other rendering options are available: anti-aliasing, shading and post processing.

Textures are essential to the vSLAM, more details in the image provide the image processing algorithms more information for feature detections. The simplest way to add a texture is to fill an object or a plane with an image. The work in [14] describes a more advanced method to create more realistic objects. Fig. 4 illustrates the texture used in the application shown in section V.

It is important to acknowledge that the bigger the quality of the textures, the higher the processing needed to render the images. A way to improve processing speed is to replicate a piece of the texture in a lower resolution along the object or plane

Fig. 4 – Texture applied in the floor of the virtual environment

IV. INTEGRATION & VISUAL ODOMETRY

This section describes the communication between MATLAB and Blender to exchange the data necessary for the vSLAM experiments.

The integration is done by text file manipulation. The robot's movement array can be written in a ".txt" file by MATLAB code and read by a Python script in Blender and vice versa. The file contains the information separated by a simple space in a specific order which is known by the writing and reading software.

In Blender for example, to read the data the code below was used:

```
rfile = open('C:\Transfer\RobotPosition.txt','r')
data = rfile.read()
x = data[3:9]
y = data[9:15]
psi= data[16:22]
rfile.close
```

With this simple code, it is possible to capture the information written on the ".txt" file in a determined period of time to generate the animation. Some additional code might be needed depending on the precision of the numbers used and if the coordinates considered are only positive and/or negative.

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 13-16 October 2015, Banff, Alberta, Canada

After making the robot follow a desired path generated by matlab, it is necessary to capture images from each or a determined number of movement steps. In the main code, a picture can be automatically generated using an existing camera on the Blender's virtual environment by adding the code:

```
bpy.context.scene.objects.active = bpy.data.objects["Camera"]
bpy.data.objects['Camera'].select = True
bpy.context.scene.camera = bpy.data.objects["Camera"]
bpy.data.scenes['Scene'].render.filepath = "C:\Transfer\Images\Camera01 - %1.0f.jpg" % (NumName)
bpy.ops.render.render( write_still = True )
bpy.data.objects['Camera'].select = False
```

The process above, consists in selecting the camera and rendering the image in its line of view and save in a determined desired path. These images are used later in an image processing algorithm to estimate the robot's translation and rotation. This methodology allows the usage of any interface able to read images and apply image processing, using the most common languages such as C++, C#, Java, Python, Matlab.

Images of video produced by Blender are sent to a specific path configured by the user. In this work, we used MATLAB to execute the image processing. The images are accessed in the path defined in the code above. Fig 5 shows a summary of the communication process.

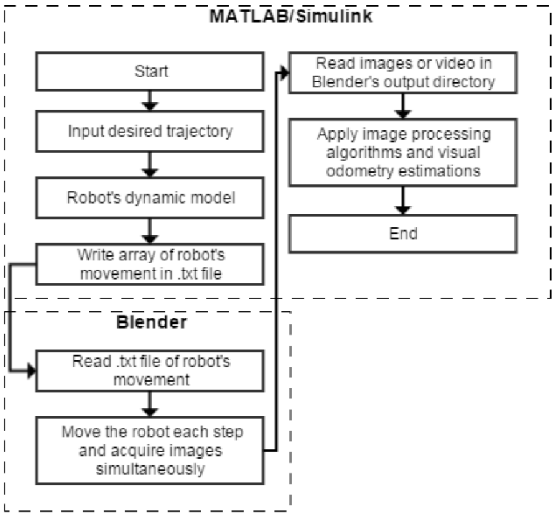


Fig. 5 – Summary of the communication process between MATLAB and Blender

Another software could be used for image processing. In example, a C# or Java application could read the Blender's output files instead of MATLAB.

V. VSLAM APLLICATION

This section describes a simple vSLAM application done in a simulated environment through the method described above.

The experiment consists in using stereoscopic images and use the optical flow technique to estimate the robot's trajectory.

The robot is programmed to follow the trajectories shown in Fig. 6. During its path, a couple of images are captured in each step of the movement and processed in MATLAB. Fig. 7 shows an example of interest points identified by the Optical Flow algorithm.

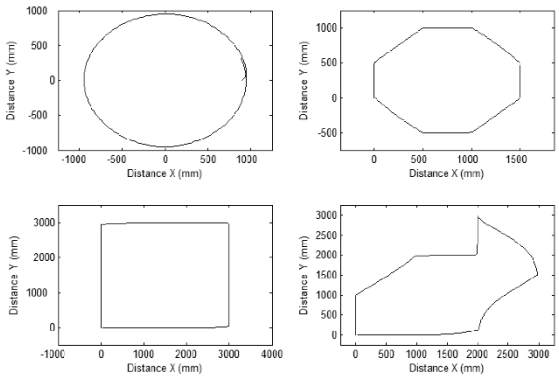


Fig. 6 – Paths used in the vSLAM experiment

Fig. 7 – Finding of interest points in left and right images

In many VO application, a type of data regularization is used. Generally, the data provided by the most used VO algorithms has a considerable imprecision and brings the need of an additional filtering step. In this work, we decided to use a Neural Network Filter. A simple network can be designed using MATLAB's Neural Fitting tool "nftool". Figure 8 illustrates the process and the characteristics of the Neural Network. The specifications used were 70% training, 15% Validation, 15% Testing, 10 hidden neurons and Bayesian Regularization. The inputs to train the network were the Circle, Octagon and Rectangle trajectories' 'x','y' and 'psi' in Figure 6 estimated by VO and the targets (output) were the known real values of such paths obtained with the robot dynamics simulation in Simulink . The concept idea was to train the network with different paths (straight, diagonal and

circular) to improve the results provided with a non-trained path.

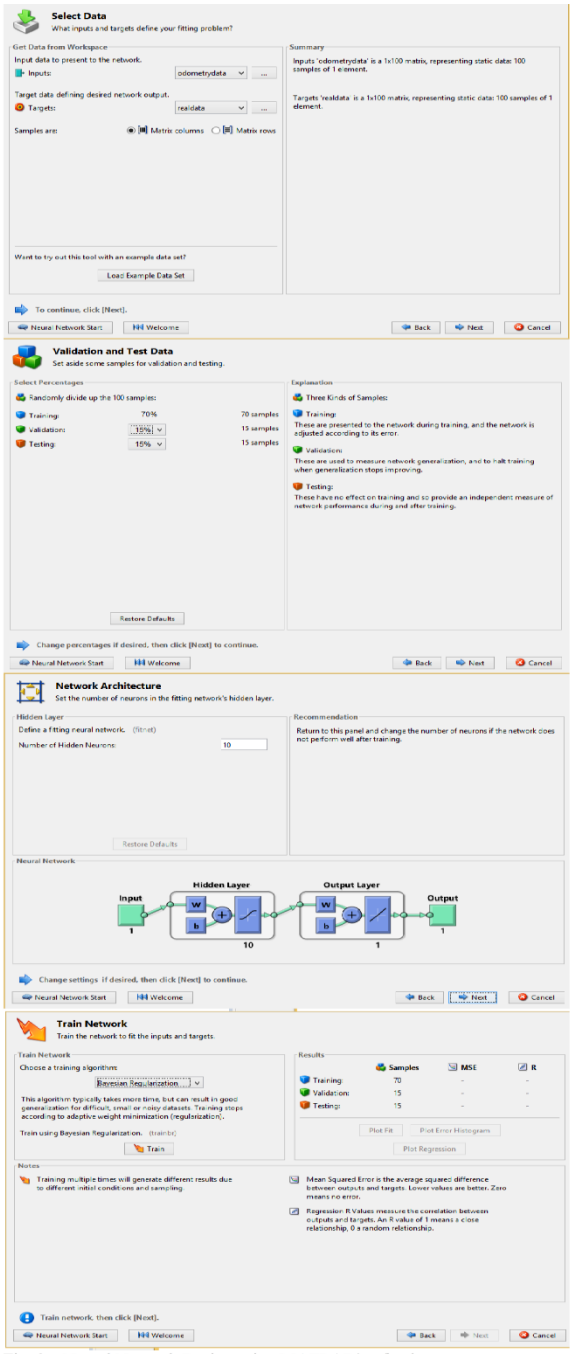


Fig. 8 – Neural Network Design using MATLAB's nftool

The results of the training are shown above in Fig. 9

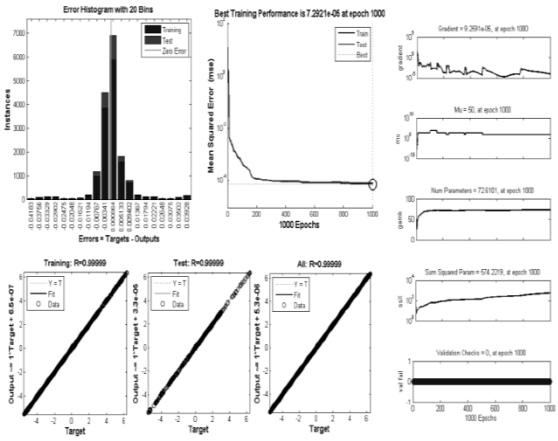


Fig. 9 – Neural Network training results

Using the information provided by the algorithm, the odometry estimation is executed and further filtering is applied to regularize the result. Fig. 10 illustrates the results obtained.

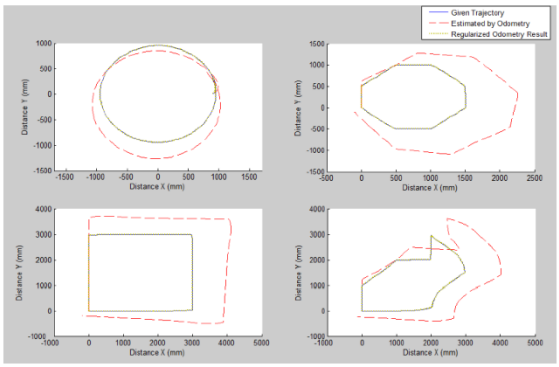


Fig. 10 – Results of the vSLAM experiment in an simulated environment

VI. Conclusion

The method described in this work can the successfully be applied in vSLAM experiments for proof of concept and even in a simulated environment with similar characteristics to the real ambient for simulations before the real experiment.

The communication between MATLAB and Blender requires some programming experience. Poor coding can lead to undesired delay in the whole process.

High quality textures provide rich information to the image processing algorithms, however processing speed is heavily affected. A lower resolution image can be used for better performance

Although the experiment present was in an indoor environment, the same methodology can be applied to outdoor ambient. Processing speed might be negatively affected

according to the number of objects and details in the environment.

Any researcher that desired fast results and easy deployment can use this method. This method provide a great way to test results before acquiring expensive equipment for vSLAM applications.

We further desire to apply the results of this work to control the trajectory of a robot in any indoor environment with minimal light conditions needed using training data to feed the Neural Network and improve the odometry results.

ACKNOWLEDGMENT

The authors wish to thank FAPEAM, the research-founding agency from the state of Amazonas, Brazil, for financing this work and the University of Campinas for providing the necessary infrastructure.

REFERENCES

[1] Weiss, Stephan, Davide Scaramuzza, and Roland Siegwart. "Monocular‐SLAM‐based navigation for autonomous micro helicopters in GPS‐denied environments." Journal of Field Robotics 28.6 (2011): 854-874.

[2] Huang, Albert S., et al. "Visual odometry and mapping for autonomous flight using an RGB-D camera." International Symposium on Robotics Research (ISRR). 2011.

[3] Konolige, Kurt, Motilal Agrawal, and Joan Sola. "Large-scale visual odometry for rough terrain." Robotics Research. Springer Berlin Heidelberg, 2011. 201-212.

[4] Sinha, U. "The Shi-Tomasi Corner Detector.", 2010.

[5] Ryu, J-B., H-H. Park, and J. Park. "Corner classification using Harris algorithm." Electronics letters 47.9 (2011): 536-538.

[6] Liu, Ce, Jenny Yuen, and Antonio Torralba. "Sift flow: Dense correspondence across scenes and its applications." Pattern Analysis and Machine Intelligence, IEEE Transactions on 33.5 (2011): 978-994.

[7] Knopp, Jan, et al. "Hough transform and 3D SURF for robust three dimensional classification." Computer Vision–ECCV 2010. Springer Berlin Heidelberg, 2010. 589-602.

[8] Sun, Deqing, Stefan Roth, and Michael J. Black. "Secrets of optical flow estimation and their principles." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[9] Brito, Allan. "Blender 3D: Jogos e animações interativas." São Paulo: Novatec (2011).

[10] Corke, Peter. "A robotics toolbox for MATLAB." Robotics & Automation Magazine, IEEE 3.1 (1996): 24-32.

[11] Martins, Felipe N., et al. "An adaptive dynamic controller for autonomous mobile robot trajectory tracking." Control Engineering Practice 16.11 (2008): 1354-1363.

[12] De La Cruz, Celso, and Ricardo Carelli. "Dynamic modeling and centralized formation control of mobile robots." *IEEE Industrial Electronics, IECON 2006-32nd Annual Conference on*. IEEE, 2006.

[13] Brito, Allan. Blender 3D: Architecture, Buildings, and Scenery: Create photorealistic 3D architectural visualizations of buildings, interiors, and environmental scenery. Packt Publishing Ltd, 2008.

[14] DANA, Kristin. Bidirectional Texture Function and 3D Texture. Computer Vision: A Reference Guide, p. 46-50, 2014.